Mein Colour-Genie

Ralf Marquis



BERKELEY · PARIS · DÜSSELDORF

Umschlagenwurf Daniel Boucherie
Satz tgr – typo-grafik-repro gmbh , remscheid
Gesamtherstellung Druckerei Hub Hoch, Dusseldorf

Der Verlag hat alle Sorgfalt walten lassen, um vollstandige und akkurate Informationen zu publizieren SYBEX-Verlag GmbH, Dusseldorf, ubernimmt keine Verantwortung fur die Nutzung dieser Informationen, auch nicht fur die Verletzung von Patent- und anderen Rechten Dritter, die daraus resultieren

ISBN 3-88745-063-9
1 Auflage 1984

Alle Rechte vorbehalten Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlages reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfaltigt oder verbreitet werden

Printed in Germany
Copyright © 1984 by SYBEX-Verlag GmbH, Dusseldorf

Inhaltsverzeichnis

Vorwort

Dieses Buch vermittelt Ihnen theoretische und praktische Kenntnisse für die Anwendung des Home-Computers COLOUR-GENIE. Sowohl die große Vielfalt der Einsatzgebiete eines solchen Rechners als auch die Fülle von BASIC-Anweisungen und Syntax-Regeln kann besonders bei Anfängern leicht zu einer gewissen Hilflosigkeit führen.

Damit dies nicht geschieht, sollten Sie Ihren Rechner von Anfang an schrittweise beherrschen lernen. In diesem Sinne ist das Buch aufgebaut. Das beim Erwerb des Colour-Genie mitgelieferte BASIC-Handbuch mit dem Titel „COLOUR BASIC – leicht gemacht" soll Ihnen parallel dazu als Nachschlageverzeichnis und Ergänzung dienen.

An dieser Stelle danken wir der Firma Trommeschläger Computer GmbH, St. Augustin, für die Unterstützung und freundliche Zusammenarbeit.

Kapitel 1

Über den Rechner

Wie viele Rechner in seiner Preisklasse verfügt das Colour-Genie über eine Reihe von Eigenschaften, die es als geradezu ideal für den Einstieg in das Computerhobby ausweisen. Eine ansprechende Tastatur, grafische und farbige Darstellungsmöglichkeiten, Tonerzeugung sowie ein umfassender Befehlsvorrat stellen nahezu optimale Ausgangsvoraussetzungen dafür dar, die Programmiersprache BASIC zu erlernen, ohne gleich an die Grenzen dieses Rechners zu stoßen. Auf diese Weise ist für nahezu jede potentielle Anwendung eine Basis geschaffen.

Bevor wir im einzelnen auf solche Rechnerkomponenten wie Prozessor, Speicher-ICs und Tongeneratorbaustein eingehen, sollen Sie einen groben Überblick darüber erhalten, welche Merkmale das Colour-Genie als solches kennzeichnen.

Das Colour-Genie gehört zur Familie der Z80-Rechner. Diese Bezeichnung ist auf den Prozessortyp zurückzuführen. Der Prozessor ist das Kernstück eines jeden Rechners. Es handelt sich um einen IC-Baustein, der in der Lage ist, mit hoher Geschwindigkeit Befehle, z. B. Rechenoperationen, auszuführen. Ein Prozessor ist das Steuerelement für jede Aktivität, die ein Rechner ausführt. Aus diesem Grunde nennt man den Prozessor auch häufig die „Zentraleinheit" oder verwendet dafür den englischen Ausdruck CPU. Die Zentraleinheit des Colour-Genie trägt die Bezeichnung Z80-CPU.

Ein weiteres Merkmal des Colour-Genie ist der BASIC-Interpreter der Firma Microsoft. Ein Interpreter ist ein Programm, das u. a. in der Lage ist, über die Tastatur eingegebene Anweisungen zu analysieren und diese an den Prozessor als Folge von codierten Anweisungen, die Maschinenspracheinstruktionen, weiterzugeben. Die gesamte Arbeit des Mikroprozessors Z80 wird nur über solche Maschinenbefehle gesteuert. Hierzu zählen Befehle wie beispielsweise:

LD	A,4	Lade den Wert 4 in den Akkumulator (Rechenwerk)
ADD	A,4	Addiere eine Zahl zum Akkumulator
JP	4000H	Verzweige zu einer vorgegebenen Speicheradresse
CP	B	Führe einen Vergleich durch

Zur Steuerung des Rechners sind also Maschinenbefehle erforderlich. Sofern für eine durchzuführende Arbeit mehrere Maschinenbefehle hintereinander vom Rechner ausgeführt werden müssen, spricht man von einer Maschinenroutine.

Der BASIC-Interpreter besteht aus einer Vielzahl solcher Maschinenroutinen. Eine elementar wichtige Routine ist die Monitor- und Bootroutine. Die erste Aufgabe dieses Interpretersegments besteht darin, den Rechner nach dem Einschalten in Millisekunden von „0 auf 100" zu bringen. In dieser Phase werden die erforderlichen Startwerte für die Tastatureingabe sowie für die Bildschirmausgabe gesetzt.

Nachdem diese Startwerte gesetzt sind, steht der Interpreter bereit zur Kontrolle der verschiedenen Bildschirm- und Tastaturoperationen. Je nach Art der über die Tastatur eingegebenen Anweisungen werden dann andere Maschinenroutinen vom Prozessor abgearbeitet: Routinen für die Verarbeitung der Eingabe, Arithmetikroutinen, Ausgaberoutinen etc.

Jeder Interpreter stellt also ein kompliziertes Gebilde von Maschinenroutinen dar, die die Fähigkeiten eines Rechners ausmachen. Der BASIC-Interpreter der Firma Microsoft ist mittlerweile auf derart viele Z80-Rechnersysteme angepaßt, daß man von einem Microsoft-Standard spricht. Microsoft-Standard heißt aber auch: Lauffähigkeit von BASIC-Programmen anderer Rechner auf dem Colour-Genie und umgekehrt.

Ein individuelles Merkmal des Colour-Genie ist sein Speicheraufbau. Hierin unterscheidet er sich erheblich von ansonsten ähnlich aufgebauten Rechnern. Neben dem Speicher, der den Interpreter beinhaltet, und dem Arbeitsspeicher, in dem BASIC-Programme abgelegt und ausgeführt werden, stehen zwei separate Bildwiederholspeicher zur Verfügung: In dem einen werden vom Rechner alle Zeichen abgelegt, die sich gegenwärtig auf dem Bildschirm befinden. Der andere kann, vollkommen getrennt vom ersten, Zeichnungen aus bunten Linien und Kurven erfassen. Zwischen beiden Bildschirmebenen kann hin- und hergeschaltet werden.

Ein weiterer Speicher steht für die Erstellung und den Abruf eigener Buchstaben und Symbole zur Verfügung.

Auf all diese Eigenschaften wird in den Folgekapiteln eingegangen.

Kapitel 2

Grafik auf dem Colour-Genie

2.1. DIE BEIDEN BILDSCHIRMDARSTELLUNGEN

Kapitel 1 konnten Sie entnehmen, daß der in Ihrem Colour-Genie zur Verfügung stehende Speicher auf verschiedene Arten genutzt wird.

Neben dem Programmspeicher gibt es einen Speicherbereich für die Bildwiedergabe von Texten und einen für hochauflösende Grafik. Diese Speicherbereiche sind im Rechner sorgfältig vom übrigen Teil getrennt, ihre Inhalte kollidieren nicht miteinander. Man nennt sie Bildwiederholspeicher. Die einzige Aufgabe solcher Bildwiederholspeicher ist es, alle Zeichen, die auf dem Bildschirm ausgegeben werden sollen, zu erfassen.

Zwischen den beiden Bildwiederholspeichern des Colour-Genies können Sie hin- und herschalten:

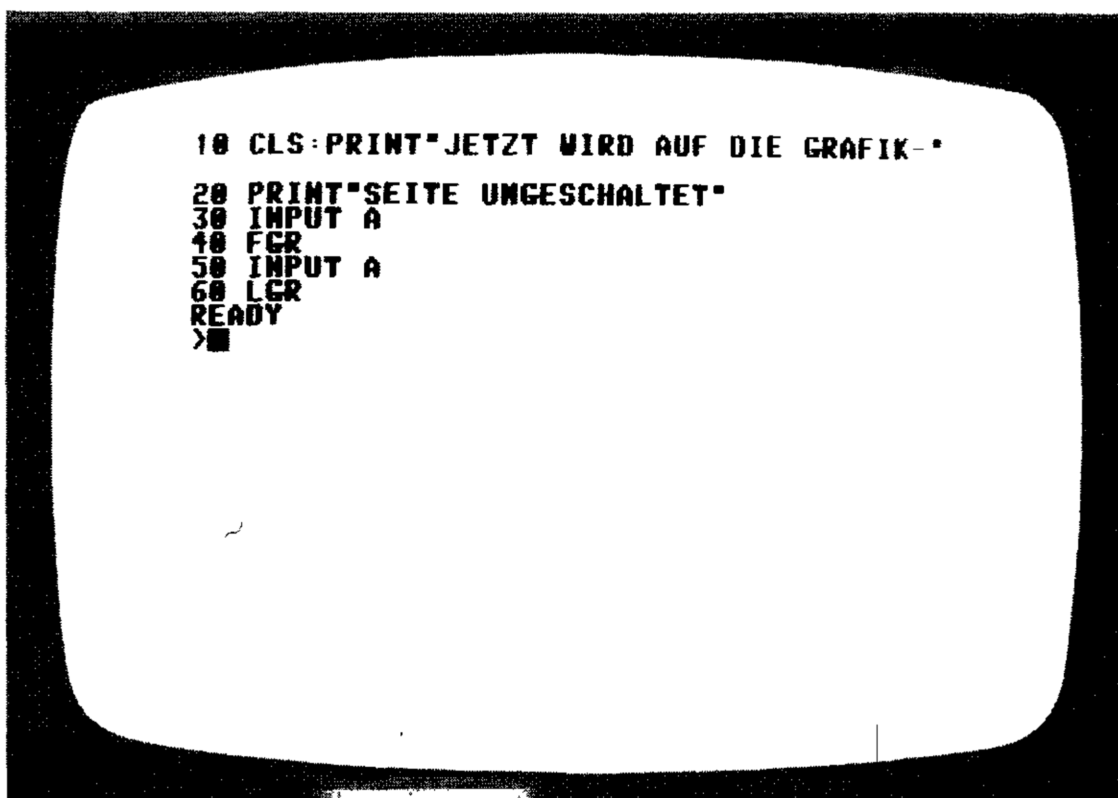


Abb. 2.1: Bildschirmumschaltung

Wenn Sie dieses Programm starten, wird die Bildschirmseite, auf der sich der Text befindet, einfach abgeschaltet. Die Umschaltung übernimmt hier die Anweisung FGR in Programmzeile 40. Aus diesem Grund nennt man die Grafikseite auch FGR-Bildschirmseite. FGR steht hier als Abkürzung für „full graphics", d. h. „Vollgrafik".

Der FGR-Bildschirm bleibt so lange eingeschaltet, bis der Rechner eine Anweisung zum Umschalten auf die Textseite vorfindet. Die Anweisung zum Umschalten auf den Textbildschirm steht in Programmzeile 60 und lautet LGR. Aus diesem Grunde nennt man die Textbildschirmseite auch LGR-Bildschirmseite.

Neben diesen beiden Anweisungen wurde eine weitere Möglichkeit geschaffen, direkt von der Tastatur aus diese Umschaltung vorzunehmen. Um die Grafikseite einzuschalten, werden die Tasten <CTRL> und <MOD SEL> gleichzeitig betätigt. Das Zurückschalten erfolgt über die Taste <BREAK>.

Jeder Bildwiederholspeicher ist für ganz bestimmte Anwendungen vorgesehen:

Der LGR-Bildschirm zeigt Zahlen, Buchstaben, frei definierte Zeichen und die Blockgrafikzeichen, die auf der Tastatur zu sehen sind. Auf dem FGR-Bildschirm steht Ihnen ein Punkteraster zur Verfügung, mit dem Sie pro Zeile 160 und pro Spalte 102 Bildpunkte darstellen können. Jeden dieser Bildpunkte können Sie über entsprechende Anweisungen einzeln ansprechen.

Leider läßt sich der Inhalt von beiden Bildwiederholspeichern nicht mischen. Auf dem Grafikbildschirm können keine Buchstaben, Ziffern, Grafik- und Sonderzeichen dargestellt werden. Hierfür benötigen Sie den LGR-Bildschirm.

Dies ist für Einsteiger häufig ein wenig verwirrend. Daher sind die einzelnen Unterschiede im folgenden noch einmal tabellarisch zusammengefaßt:

	LGR-Bildschirm	FGR-Bildschirm
Anzahl der möglichen Farben	16	4
Anzahl der frei definierbaren Zeichen	128	keine
Sind Bildpunkte einzeln ansteuerbar?	nein	ja
Kann man Linien und Kreise zeichnen?	nein	ja
Können Flächen farbig ausgemalt werden?	nein	ja
Kann Text dargestellt werden?	ja	nein
Können Grafikzeichen dargestellt werden?	ja	nein

2.2. FARBIGE DARSTELLUNG IM TEXTBILDSCHIRM

Wie Sie der vorigen Tabelle entnehmen können, verfügt das Colour-Genie über 16 mögliche Farben auf dem LGR-Bildschirm. Zum Anwählen einer bestimmten Farbe dient die COLOUR-Anweisung. Alle Zeichen, die nach einer COLOUR-Anweisung auf dem Bildschirm ausgegeben werden, erscheinen in der gewählten Farbe:

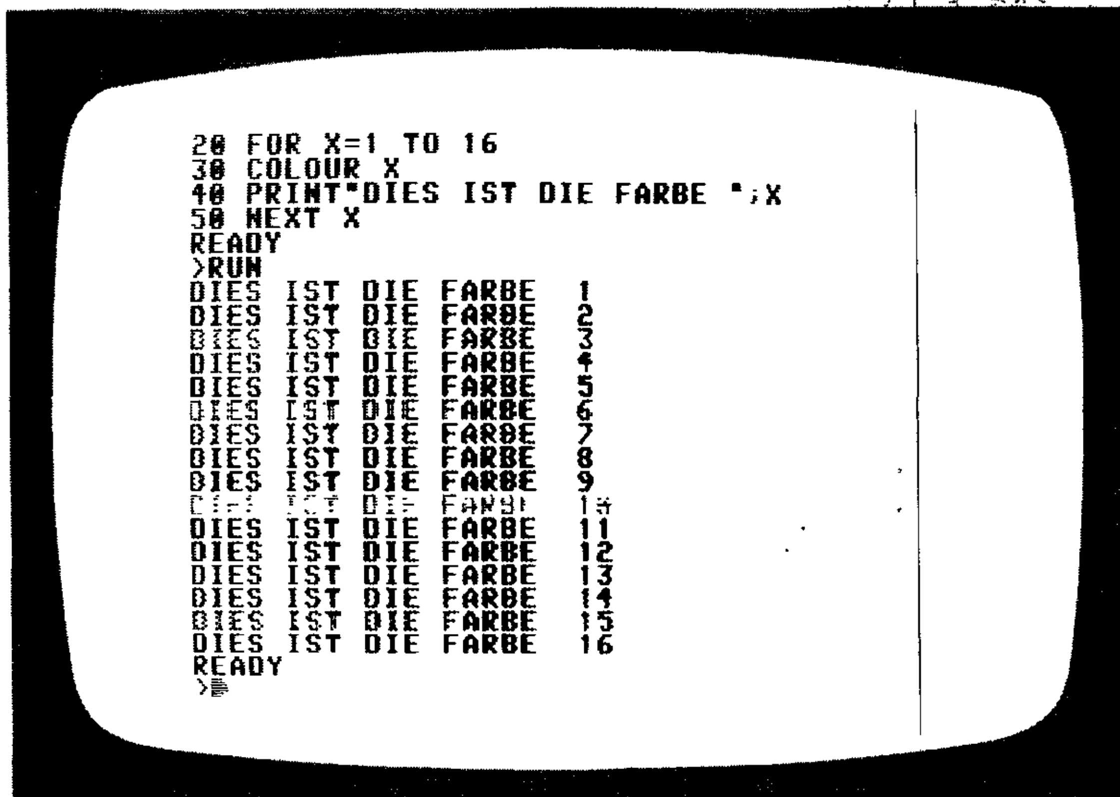


Abb. 2.2: Farbauswahl

Wenn Sie mit Programmen arbeiten, bei denen die Farbe auf dem Bildschirm von Bedeutung ist, sollten Sie zuvor kurz dieses Programm eingeben, um Ihren Farbfernseher so einzustellen, daß sich ein sauberes Farb- und Buchstabenmuster auf dem Bildschirm ergibt. Dies ist ziemlich mühsam, da die Bandbreite des Modulators, der sich in Ihrem Colour-Genie befindet und der das Fernsehbild erzeugt, sehr klein ist, und somit die Einstellung ein wenig Fingerspitzengefühl erfordert.

Haben Sie den LGR-Bildschirm justiert, können Sie sich gleich einmal alle Zeichen darauf ausgeben lassen, die das Colour-Genie kennt:

```
20 FOR X=33 TO 255
30 PRINT"ZEICHEN NR.:";X,CHR$(X)
40 NEXT
READY
>RUN
ZEICHEN NR.: 33      !
ZEICHEN NR.: 34      "
ZEICHEN NR.: 35      #
ZEICHEN NR.: 36      $
ZEICHEN NR.: 37      %
ZEICHEN NR.: 38      &
ZEICHEN NR.: 39      '
ZEICHEN NR.: 40      (
ZEICHEN NR.: 41      )
ZEICHEN NR.: 42      *
ZEICHEN NR.: 43      +
ZEICHEN NR.: 44      ,
ZEICHEN NR.: 45      -
ZEICHEN NR.: 46      .
ZEICHEN NR.: 47      /
ZEICHEN NR.: 48      0
ZEICHEN NR.: 49      1
```

Abb. 2.3: Der Standardzeichensatz des Colour-Genies

```
20 X=RND(16) 'X IST EINE ZUFALLS-
              ZAHL ZWISCHEN 1 UND 16
30 COLOUR X
40 PRINT@400,"DIESER TEXT WECHSELT SEINE
 FARBE."
50 GOTO 20
READY
>RUN

DIESER TEXT WECHSELT SEINE FARBE
```

Abb. 2.4: Zufallsfarben

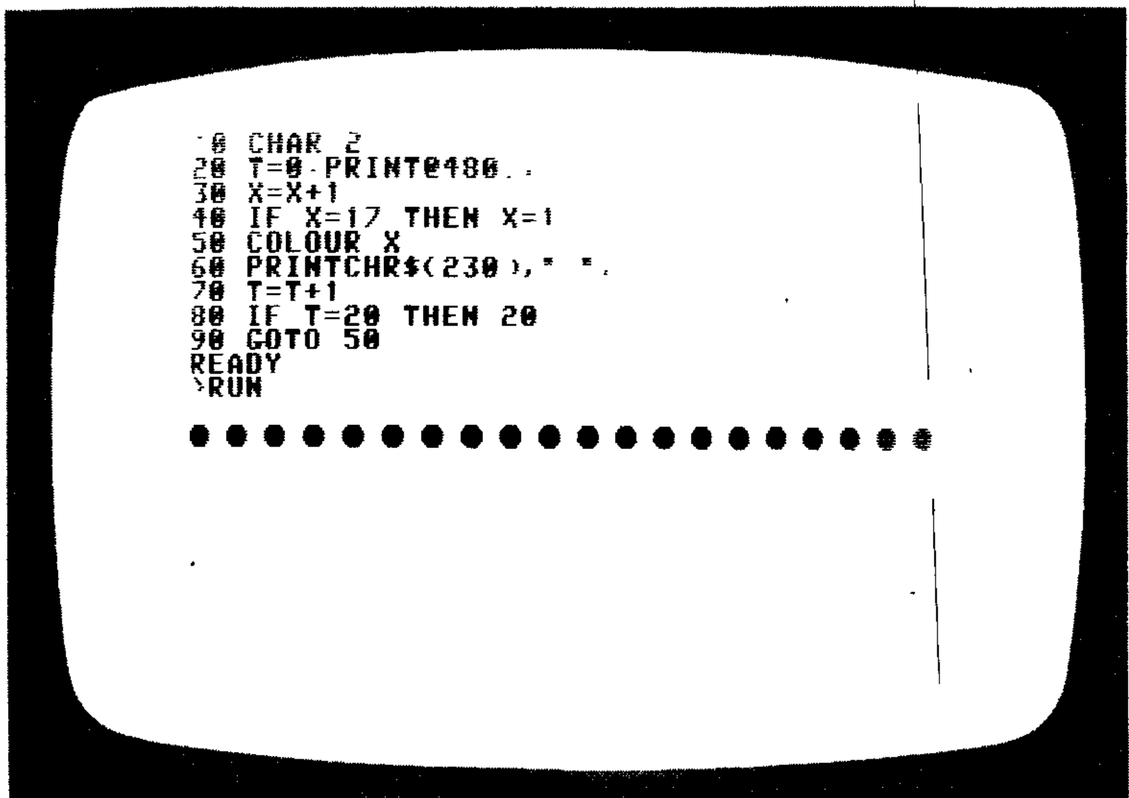


Abb. 2.5: Lauflicht

Diese Zeichen sind auch in Ihrem BASIC-Handbuch auf den Seiten 123 und 147 zu finden.

Einen interessanten optischen Effekt erzielen Sie, indem Sie einen bestimmten Text immer an der gleichen Bildschirmstelle in ständig wechselnden Farben ausgeben lassen:

Auch der „Lauflichteffekt“, wie man ihn aus Diskotheken kennt, läßt sich auf dem Bildschirm nachvollziehen:

Neben der Möglichkeit, Farben über die COLOUR-Anweisung anzuwählen, kann dies auch direkt über die Tastatur geschehen. Dazu betätigen Sie die <CTRL>-Taste zusammen mit einer der Tasten <1> bis <8>. Die Farbe, die der betätigten Taste entspricht, ist auf der Tastatur vermerkt. Alle Folgezeichen werden vom Rechner in der gewählten Farbe ausgegeben.

2.2.1 Die Blockgrafikzeichen auf der Tastatur

Neben den „normalen“ Zeichen einer Schreibmaschinentastatur verfügt das Colour-Genie über Blockgrafikzeichen, die durch Betätigung der <MOD SEL>-Taste angesprochen werden können. Die <MOD SEL>-Taste dient hierbei als Schalter zwischen den „normalen“ und den Grafikzeichen:

PRINT "a b (tippen Sie hier <MOD SEL>) a b (tippen Sie hier <MOD SEL>) a b"

Durch die eingebauten Grafikzeichen wird es möglich, im LGR-Bildschirm Grafik und Text miteinander zu mischen. Sie können z. B. den gesamten Bildschirm umrahmen:

```
10    CLS
20    F$=CHR$(230)
30    FOR X=39 TO 940 STEP 40
40    PRINT@X,F$;:PRINTF$;
50    NEXT X
60    FOR X=0 TO 39
70    PRINT@X,F$;
80    PRINT@920+X,F$;
90    NEXT X
100   PRINT@410,"DER RAHMEN";
110   GOTO 110
```

Abb. 2.6: Beispielprogramm für Blockgrafik

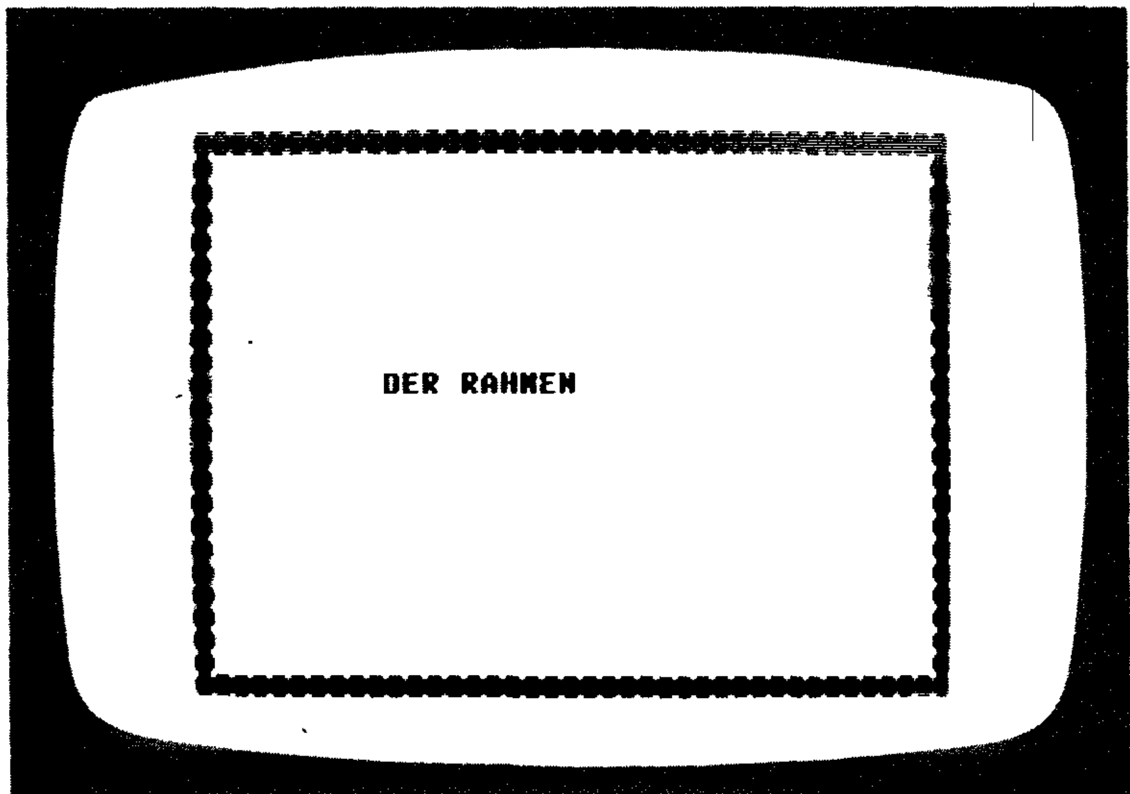


Abb. 2.7: Bildschirmausgabe – ein Rahmen in Blockgrafik

Jedes Blockgrafikzeichen besitzt übrigens einen ganz individuellen sog. ASCII-Code, den Sie auf Seite 147 Ihres Handbuches nachlesen können. Statt der Tastenkombination <MOD SEL> <a>, die das Grafikzeichen ● auf dem Bildschirm darstellt, kann auch der entsprechende ASCII-Code des gewünschten Grafikzeichens eingegeben werden. Dies geschieht in Programmzeile 20 durch die Zuweisung F$ = CHR$ (230).

In den Blockgrafikzeichen unterscheidet sich das Colour-Genie erheblich von ansonsten ähnlich aufgebauten Rechnern wie dem Video Genie und dem TRS 80.

Blockgrafikzeichen können Sie auf zwei Arten ansprechen:

1. Über die Tastatur in Verbindung mit der <MOD SEL>-Taste
2. Mit Hilfe der CHR$-Funktion, indem Sie aus der ASCII-Codetabelle die Nummer des gewünschten Zeichens angeben.

Beide Varianten haben ihre Vor- und Nachteile. Mit der ersten Variante sollten Sie dann arbeiten, wenn Sie mit vielen Blockgrafikzeichen auf einmal arbeiten und diese schon beim Auflisten des Programms einen Eindruck darüber vermitteln sollen, wie die spätere Grafik aussieht.

Die zweite Variante ist dann empfehlenswert, wenn Programmlistings auf dem Drucker erstellt werden sollen. Matrixdrucker verfügen in der Regel über einen anderen Zeichensatz als das Colour-Genie, so daß auf dem Drucker ausgegebene Blockgrafikzeichen anders aussehen als die auf dem Bildschirm. Fazit:

Es hat vom Programmablauf her dieselbe Wirkung, ob Sie

PRINT“●“

oder

PRINT CHR$(230)

eingeben. Soll das Programm jedoch auf einem Drucker gelistet werden, ist die letzte Variante zu empfehlen.

Wenn Sie die Blockgrafikzeichen direkt über die Tastatur eingeben, werden Sie feststellen, daß diese auf der Tastatur oft ganz anders wirken als auf dem Bildschirm.

Einzelne Grafiksymbole scheinen sich kaum voneinander zu unterscheiden und passen auch irgendwie nicht zusammen. Sie sollten daher Grafikzeichen in Typengruppen einteilen. Am einfachsten erstellen Sie sich eine Tabelle, in der Sie alle Grafikzeichen in Gruppen klassifizieren:

1. Symbole (Würfel, Herz, Pik, Kreuz, Karo)
2. Linienmuster (dick, dünn, einfach, zweifach, waagerecht, senkrecht, schräg)
3. Blöcke (gestrichelt, gepunktet, etc.)
4. Kreise
5. Sonstige

Wenn Sie sich eine solche Liste anfertigen, können Sie genau sehen, welche Grafikzeichen zueinander passen und welche nicht. Dies erleichtert Ihnen erheblich die Arbeit, wenn es wie im nächsten Kapitel darum geht, größere Figuren, Zeichnungen oder Skizzen zu erstellen. Das Ganze können Sie experimentell auf dem Bildschirm nachvollziehen, indem Sie bestimmte Symbole hinter- und untereinander setzen.

Blockgrafikzeichen können nicht nur nebeneinander, sondern auch untereinander gesetzt werden. Um mehrere Blockgrafikzeichen untereinander zu setzen, d. h. um das folgende Zeichen eine Zeile tiefer zu setzen, dürfen Sie nicht die <RETURN>-Taste betätigen, denn das wird vom Rechner als Aufforderung aufgefaßt, einen (wie der Rechner glaubt) Befehl auszuführen. Das Ergebnis wäre ein „Syntax Errr“.

Um dies zu vermeiden, sollten Sie jedesmal, wenn Sie in eine neue Zeile gehen wollen, den <Pfeil nach unten> verwenden.

Der Aufwand für eine derartige Tabelle ist recht mühsam. Er ermöglicht es jedoch, sich ein Bild darüber zu machen, was auf dem LGR-Bildschirm grafisch darstellbar ist und was nicht. Sie können dann später mit erheblich weniger Zeitaufwand Figuren und grafische Skizzen erstellen. Im folgenden Abschnitt werden wir einige Beispiele hierfür sehen.

2.2.2 Zeichnungen aus Blockgrafikzeichen

Schon im vorigen Abschnitt ist Ihnen sicherlich bei dem Versuch, Figuren oder einfach eine größere Anordnung von grafischen Symbolen auf dem Bildschirm darzustellen, aufgefallen, daß das Erstellen von Zeichnungen mit Schwierigkeiten verbunden und bestimmten Beschränkungen unterworfen ist. Grund für diese Beschränkungen ist in erster Linie das Ärgernis, daß Sie nicht über die Pfeiltasten den Cursor an jede beliebige Bildschirmstelle positionieren können. So ist es weder mit der Cursor-Taste <Pfeil hoch> noch mit einer <Shift>-Kombination möglich, von einer bestimmten Bildschirmzeile in die nächsthöhere zu gelangen. Das hat zur Folge, daß z. B. Korrekturen am bereits bestehenden Teil einer Zeichnung nicht durchgeführt werden können. Es gibt keine Taste auf der Tastatur, die das Steuerzeichen mit dem ASCII-Code 27 liefert, das den Cursor eine Zeile höher positioniert. Dazu muß ein Programm geschrieben werden, das den Pfeiltasten neue Bedeutungen zuweist:

```
10    CLS
20    COLOUR 1
30    PRINT CHR$(14);
40    Y$=INKEY$:IFY$=""THEN40
50    IFY$=CHR$(13)THENY$=CHR$(29)+CHR$(26)
60    IFY$=CHR$(91)THENY$=CHR$(27)
70    IFY$=CHR$(10)THENY$=CHR$(26)
80    IFY$=CHR$(9)THENY$=CHR$(25)
90    IFY$=CHR$(8)THENY$=CHR$(24)
100   PRINTY$;:GOTO40
```

Abb. 2.8: Cursorsteuerung per Programm

Dieses Programm löscht zuerst den Bildschirm. Dann erscheint in der linken oberen Bildschirmecke ein blinkender Cursor. Sie können nun über die Pfeiltasten auf der Tastatur den Cursor an eine von Ihnen gewählte Bildschirmposition bringen und so den Bildschirm beliebig mit Zeichen füllen. Wenn Sie z. B. den Linkspfeil betätigen, wird nicht mehr wie bisher das zuletzt eingegebene Zeichen gelöscht, sondern nur der Cursor eine Position nach links gesetzt.

Man nennt solche ASCII-Zeichen, durch die kein Zeichen auf dem Bildschirm ausgegeben, sondern nur die Position des Cursors geändert oder Teile des Bildschirms gelöscht werden, Bildschirmsteuerzeichen. Eine Liste dieser Bildschirmsteuerzeichen und ihrer Bedeutungen finden Sie auf Seite 145 Ihres BASIC-Handbuchs. Welche zusätzlichen Anwendungsmöglichkeiten Bildschirmsteuerzeichen bieten, wird später im Kapitel 2.4 eingehend erläutert.

Durch diese Cursor-Steuerungsroutine sind Sie also in der Lage, einen kompletten Bildschirm in beliebiger Reihenfolge mit Zeichen zu füllen. Wenn die <MOD SEL>-Taste betätigt wird, können dies natürlich auch Blockgrafikzeichen sein. Die hier aufgeführte Routine ist ein häufiger Bestandteil von Textverarbeitungsprogrammen.

Sie können nun Skizzen, Grafiken oder andere Zeichnungen erstellen. Ein Beispiel finden Sie am Ende dieses Abschnitts.

In der Regel wollen Sie sicherlich eine einmal erstellte Skizze oder Zeichnung weiterverarbeiten können. Dazu muß die Möglichkeit geschaffen werden, alles, was sich auf dem Bildschirm befindet, irgendwie abzuspeichern. Um dies zu bewerkstelligen, machen wir uns spezielle Anweisungen des Microsoft-BASICs zunutze:

Im BASIC des Colour-Genie gibt es eine Funktion VARPTR. Diese Funktion liefert uns zu jeder gewünschten Variablen eines sich im Speicher befindenden Programms einen Hinweis, wo sich die zu dieser Variablen gehörenden Daten befinden.

Haben wir z. B. gesagt:

A$="HASE"

so erhielt der Rechner die Anweisung, der Variablen A$ einen Wert zuzuweisen, nämlich die Zeichenfolge „HASE". Diese Zeichenfolge wurde vom Rechner an einer bestimmten Stelle im Speicher abgelegt. Die Ablageadresse und weitere Hinweise, die die Variable A$ betreffen, „merkt" sich der Rechner.

Durch Eingabe von

C=VARPTR(A$)

kann man sich diese Informationen ausgeben lassen. So liefert diese Anweisung zunächst einmal einen Hinweis, daß in Speicherstelle C die Länge von A$ steht (in diesem Falle eine 4).

In den Speichersteellen C+1 und C+2 steht dann ein Hinweis, wo wir die Zeichenkette A$ im Speicher finden können. Wir nutzen die Informationen, die wir durch VARPTR erhalten. Wir können dann später jeden beliebigen Bildschirminhalt als Zeichenkette an irgendwelche Programmvariablen übergeben. Zunächst ein Beispiel, wie VARPTR genutzt werden kann:

```
10    A$=" "
20    AD=VARPTR(A$)
30    AB=PEEK(AD+1)+256*PEEK(AD+2)
40    IF AB>32767 THEN AB=AB-65536
50    POKE AB,ASC("B")
```

Abb. 2.9: Die Verwendung von VARPTR

Wenn Sie dieses Programm starten und anschließend

LIST

eingeben, werden Sie feststellen, daß sich Programmzeile 10 verändert hat, da die Variable A$ nun den Wert "B" besitzt.

Mit Hilfe der VARPTR-Funktion wurde in Zeile 30 die Speicheradresse ermittelt, wo der Rechner A$=" " im Speicher abgelegt hatte. Diese

Wertzuweisung wurde verändert, indem direkt in die ermittelte Speicheradresse durch die Anweisung in Zeile 50 ein anderer Wert, nämlich "B" eingetragen wurde.

2.2.3 Ein Grafikeditor

Um einen ganzen Bildschirminhalt in Variablen zu erfassen, sind 1000 Bytes notwendig, da der LGR-Bildschirm eine Auflösung von 25 Zeilen mit je 40 Zeichen besitzt. Diese 1000 Bytes können bequem in 5 Zeichenkettenvariablen à 200 Bytes erfaßt werden.

Für alle 1000 Zeichen, die aus dem Bildschirm ausgelesen werden sollen, muß von vornherein im Programm Platz geschaffen werden. Dies geschieht derart, daß den 5 Zeichenkettenvariablen jeweils 200 Zeichen zugewiesen werden, z. B. durch die Zuweisung:

```
7 Z$(1)="****************************************************
*************************************************************
*************************************************************
*******************************************"
```

Ob Sie diese Zeichenketten nun mit Leerzeichen oder mit anderen ASCII-Zeichen füllen, ist irrelevant. Nach dem Auslesevorgang sind die alten Werte auf jeden Fall mit neuen überschrieben.

Die Vorteile, die Ihnen ein solcher Grafikeditor bietet, liegen klar auf der Hand: Nach dem Auslesevorgang verfügen Sie über fünf Variablen, die eine komplette LGR-Bildschirmseite beinhalten, und dies gleich als Programmzeilen. Wenn Sie alle Programmzeilen außer 7, 9, 11, 13 und 15 löschen, können Sie diesen Bildschirm separat weiterverarbeiten.

Zur Ausgabe eines Bildschirms werden dann nur zwei weitere Programmzeilen benötigt:

```
17 Z$(5)=LEFT$(Z$(5),199):FOR X=1 TO 5:PRINT Z$ (X);:NEXT X
18 GOTO 18
```

Es folgt nun ein Programm, mit dem Sie auf dem Bildschirm beliebige grafische Entwürfe machen können. Übernehmen Sie bitte zunächst dieses Programm in Ihren Rechner.

```
1      CLEAR 1500
2      CLS:PRINT"PROGRAMM ZUM ERSTELLEN EINER
       GRAFIKSEITE AUF DEM BILDSCHIRM. DIESE WIRD D
       URCH BETAETIGUNG VON

       <F1> DIREKT IN DEN VARIABLEN Z$(1) ... Z$(5)
        GESPEICHERT.
```

Abb. 2.10: Grafikeditor

```
        <F2> BEENDET DEN PROGRAMMLAUF.
3       INPUT"BITTE <RETURN>";AA$
4       DIM Z$(5)
5       FL=0
6       REM JEDES Z$(X) MUSS GENAU 200 ZEICHEN LANG
        SEIN. UEBERPRUEFUNG JEWEILS IN DER FOLGEZEIL
        E. IN DEN EINZELNEN Z$(X) STEHT NACH BETAETI
        GUNG DER <F1>-TASTE DER  KOMPLETTE BILDSCHIR
        MINHALT.
7       Z$(1)="*************************************
        ********************************************
        ********************************************
        ********************************************
        *******************************"
8       P=LEN(Z$(1)):IF P<>200 THENPRINT"Z$(1) IST U
        NZULAESSIG LANG: ";P;"ZEICHEN":FL=1
9       Z$(2)="*************************************
        ********************************************
        ********************************************
        ********************************************
        *******************************"
10      P=LEN(Z$(2)):IFP<>200 THENPRINT"Z$(2) IST UN
        ZULAESSIG LANG: ";P;"ZEICHEN":FL=1
11      Z$(3)="*************************************
        ********************************************
        ********************************************
        ********************************************
        *******************************"
12      P=LEN(Z$(3)):IF P<>200THENPRINT"Z$(3) IST UN
        ZULAESSIG LANG : ";P;"ZEICHEN LANG":FL=1
13      Z$(4)="*************************************
        ********************************************
        ********************************************
        ********************************************
        *******************************"
14      P=LEN(Z$(4)):IFP<>200THENPRINT"Z$(4) IST UNZ
        ULAESSIG LANG :";P;"ZEICHEN":FL=1
15      Z$(5)="*************************************
        ********************************************
        ********************************************
        ********************************************
        *******************************"
16      P=LEN(Z$(5)):IFP<>200THENPRINT"Z$(5) IST UNZ
        ULAESSIG LANG :";P;"ZEICHEN":FL=1
17      IF FL=1 THENPRINT"KORREKTUR NOTWENDIG":STOP
18      REM HIER BEGINNT DIE TASTATURABFRAGEROUTINE
19      CLS:PRINTCHR$(14);
20      Y$=INKEY$:IFY$=""THEN20ELSEA=ASC(Y$)
21      IF PEEK(-2040)=16THEN30:'<F1> ??
22      IF PEEK(-2040)=32 THEN 41:'<F2>??
23      IF(A=13)PRINTCHR$(29);CHR$(26);:GOTO20:'DIE
        <RETURN>-TASTE WIRD UMDEFINIERT
24      IF(A=91)LETA=27:'PFEIL HOCH WIRD UMDEFINIERT

25      IF(A>=32)PRINTCHR$(A);:GOTO20
26      IF A>=24 THEN 28
```

Abb. 2.10: Grafikeditor (Fortsetzung)

```
27    A=A+16:'ALLE TASTEN, DEREN ASCII-WERT <24 IS
      T, WERDEN UMDEFINIERT
28    PRINTCHR$(A);:GOTO20
29    REM HIER BEGINNT DIE BILDSCHIRMAUSLESEROUTIN
      E
30    FOR X=0 TO 4
31    G=VARPTR(Z$(X+1))
32    IF X=0THEN G=G+7
33    A1=PEEK(G+1)+256*PEEK(G+2):'AB ADRESSE A1 LI
      EGT DER TEXTSPEICHER FUER DIE VARIABLE Z$(X+
      1)
34    BS=17408+X*200
35    FOR T=0 TO 199
36    POKE A1+T,PEEK(BS+T)
37    T1=PEEK(BS+T):POKEBS+T,246:POKEBS+T,T1:'BLIN
      KENDER PUNKT AUF DEM BILDSCHIRM
38    NEXT T
39    NEXT X
40    GOTO19
41    Z$(5)=LEFT$(Z$(5),199):'WENN NICHT DAS LETZT
      E ZEICHEN DES LETZTEN STRINGS UM 1 GEKUERZT
      WIRD, SCROLLT DER BILDSCHIRM
42    CLS:PRINT:PRINT"BILDSCHIRM ABGESPEICHERT.

      JEWEILS 5 BILDSCHIRMZEILEN MIT JE 40
      ZEICHEN SIND UEBER DIE VARIABLEN
43    PRINT"
      Z$(1) => ZEILE 1 BIS 5
      Z$(2) => ZEILE 6 BIS 10
      Z$(3) => ZEILE 11 BIS 15
      Z$(4) => ZEILE 16 BIS 20
      Z$(5) => ZEILE 21 BIS 25
44    PRINT"
      ABGESPEICHERT UND KOENNEN DURCH EINGABE
      VON

      FORX=1TO5:PRINTZ$(X);:NEXTX

      JEDERZEIT AUFGERUFEN WERDEN.
45    PRINT"<RETURN> ZUM ANZEIGEN DES ABGESPEICHER
      TEN BILDES"
46    INPUTAA$
47    CLS:FOR X=1 TO 5
48    PRINTZ$(X);
49    NEXT X
50    GOTO50
```

Abb. 2.10: Grafikeditor (Fortsetzung)

Anmerkungen zum Programm:

1. Eine komplette Bildschirmseite steht Ihnen wie ein Zeichenbrett zur Erstellung eines Bildes zur Verfügung.

2. Wenn die Zeichnung erstellt ist und die Funktionstaste <F1> betätigt wurde, wird über eine spezielle Routine des Programms jedes einzelne

Zeichen auf dem Bildschirm ausgelesen und in einer der zur Verfügung stehenden Zeichenkettenvariablen gespeichert. Dieser Vorgang nimmt etwas Zeit in Anspruch. Ein kleiner blinkender Fleck zeigt Ihnen jedoch immer das zuletzt ausgelesene Zeichen auf dem Bildschirm an.

3. Nach einmaligem Auslesen kann an der Zeichnung weitergearbeitet werden. Zu diesem Zeitpunkt haben jedoch die Variablen Z$(1) bis Z$(5) bereits den ganzen Bildschirminhalt gespeichert, der sich zum Zeitpunkt des letzten Auslesens auf dem LGR-Schirm befand.

4. Ist die Zeichnung fertig, können Sie durch Betätigung der Taste <F2> weitere Hinweise abrufen. Der Inhalt der Variablen Z$(1) bis Z$(5) wird anschließend auf dem Bildschirm dargestellt.

Alle Zeichen, die sich zum Zeitpunkt der Betätigung der Funktionstaste <F1> auf dem Bildschirm befanden, sind nun in der Zeichenkettenvariablen Z$(1) bis Z$(5) von Programmzeile 7 bis Programmzeile 15 untergebracht. Sie können sich durch Eingabe von

```
LIST
```

anschauen, ob das Programm ordnungsgemäß ablief:

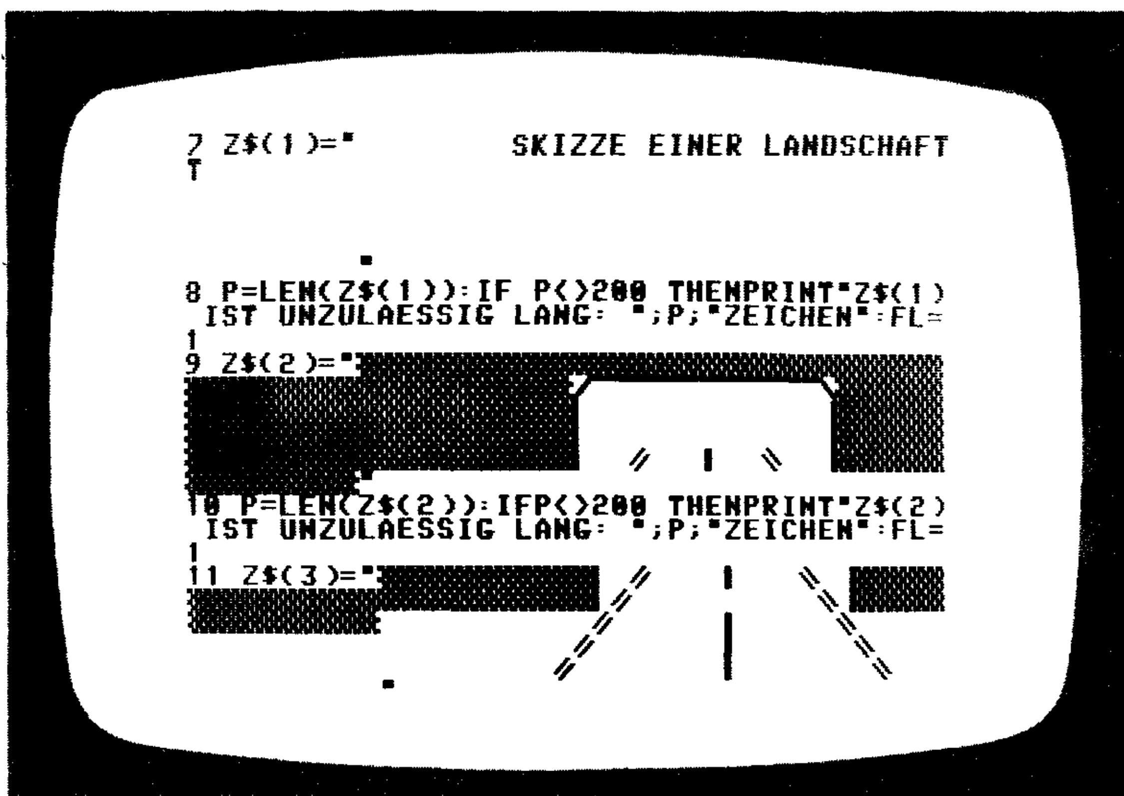


Abb. 2.11: Teil der Programmliste nach Programmende

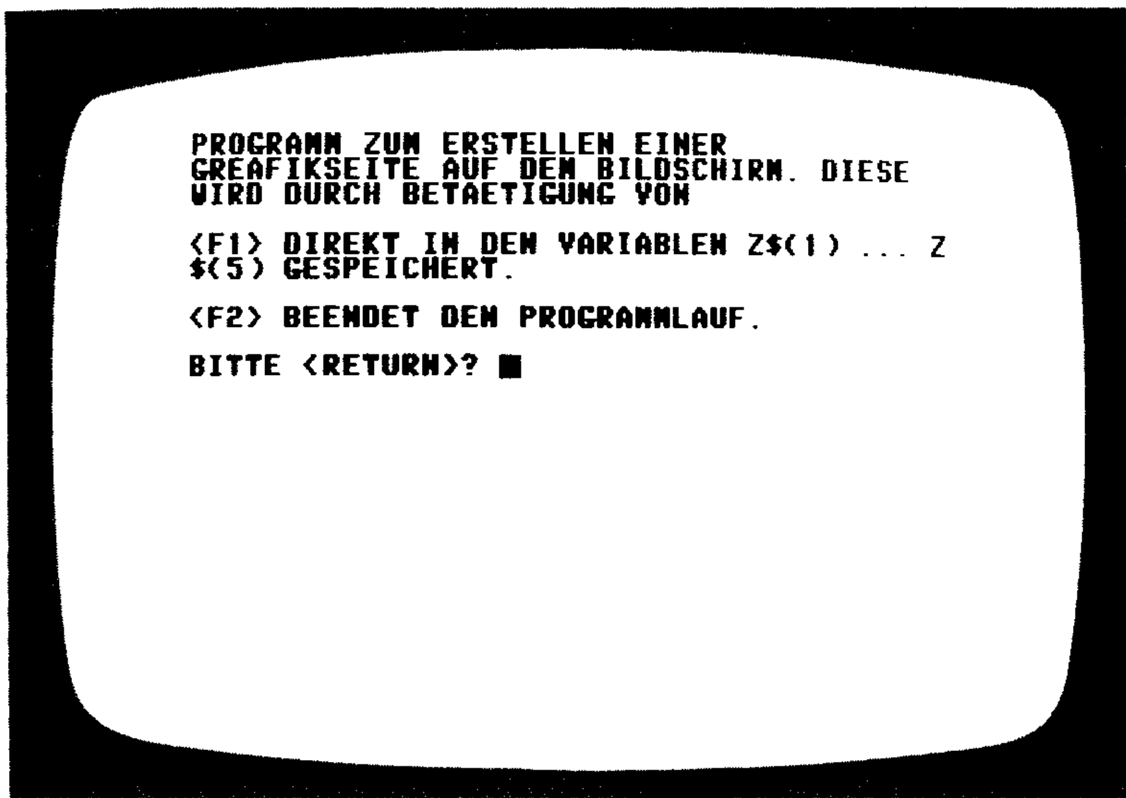


Abb. 2.12: Einleitungstext zum Grafikeditor

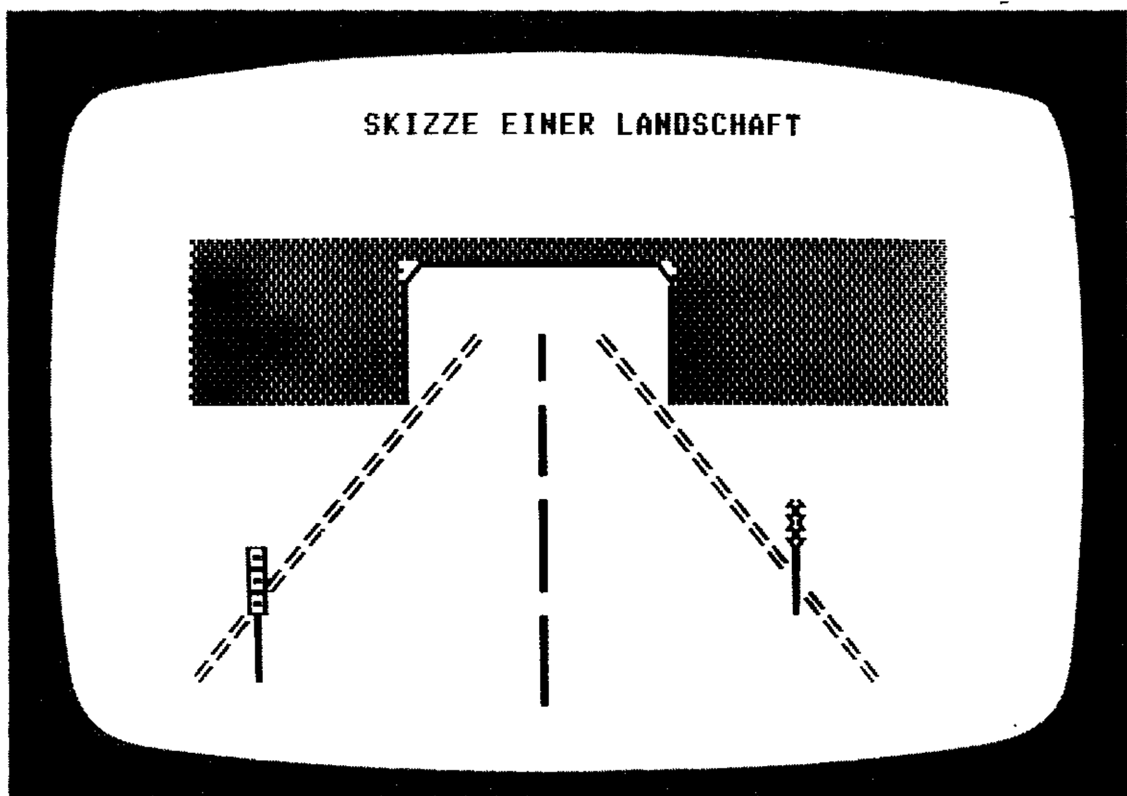


Abb. 2.13: Eine mit dem Grafikeditor erstellte Skizze

Nachdem Sie das Programm eingetippt und mit RUN gestartet haben, erscheint auf dem Bildschirm der in Abb. 2.12 dargestellte Text.

Wenn Sie nun die <RETURN>-Taste betätigen, wird der Bildschirm gelöscht. Links oben erscheint ein blinkender Cursor und zeigt die momentane Eingabeposition. Sobald Sie Ihr Bild erstellt haben, betätigen Sie die Funktionstaste <F1> zum Abschluß. In Abb. 2.13 finden Sie das Beispiel einer mit dem Grafikeditor erstellten Skizze.

Nun können Sie, wenn Sie wollen, den Grafikeditor löschen, indem Sie nur die Zeilen stehenlassen, in denen sich die Variablen Z$(1) bis Z$(5) befinden:

```
DELETE 16–50: DELETE 1–6: DELETE 8: DELETE 10: DELETE 12:
DELETE 14
```

Wenn Sie wollen, können Sie nun das Bild, das sich in den Programmzeilen 7 bis 15 befindet, mit dem Befehl

```
SAVE"(Bildname)"
```

auf Diskette oder mit

```
CSAVE"(Bildname)"
```

auf Kassette speichern.

2.2.4 Das Arbeiten mit frei definierbaren Zeichen

Eine weitere Form der Darstellung von Zeichen auf dem LGR-Bildschirm sind die frei programmierbaren Zeichen. Im Rechnerspeicher Ihres Colour-Genie wurde eigens ein Speicherbereich für die Erfassung dieser Zeichen reserviert. Hier können bis zu 128 solcher Zeichen abgelegt werden.

Da nach dem Einschalten des Computers noch keine Zeichen programmiert sind, gibt der Rechner beim Abruf eines solchen Zeichens ein Leerzeichen aus. Lassen Sie uns versuchen, das erste der frei programmierbaren Zeichen abzurufen:

```
CHAR 1:PRINT CHR$(128)
```

Mit der CHAR-Anweisung schaltet der Rechner auf einen anderen Zeichensatz um, den Zeichensatz 1. Durch diese Umschaltung werden die frei programmierbaren Zeichen zur Ausgabe auf den Bildschirm freigegeben.

Das erste Zeichen der ersten Gruppe frei programmierbarer Zeichen trägt die ASCII-Codenummer 128. Falls Ihnen der CHAR-Befehl nicht vertraut sein sollte, lesen Sie bitte unter Kap. 3.3 im BASIC-Handbuch Einzelheiten darüber nach.

Da wir zu diesem Zeitpunkt noch kein frei programmierbares Zeichen definiert haben, wird statt dessen ein Leerzeichen ausgegeben. Das erste der frei programmierbaren Zeichen belegt also in der ASCII-Tabelle die Codenummer 128.

Wir wollen nun den Buchstaben X neu entwerfen und als Zeichen mit der Nummer 128 ablegen. Schon oben wurde erwähnt, daß es sich bei dem Zeichen mit dieser Codenummer um das erste der frei programmierbaren Zeichen handelt. Aus diesem Grunde liegen die Speicheradressen, die für dieses Zeichen zuständig sind, ganz am Anfang des Speicherbereichs, der im Rechnerspeicher für die frei programmierbaren Zeichen reserviert wurde. Im Speicherbelegungsplan auf Seite 124 Ihres Handbuchs steht genau, welche Adresse dieser Speicher belegt:

F400 bis F7FF

Das entspricht dezimal den Adressen −3072 bis −2049. Und so sieht der Speicher für das erste Zeichen anfangs aus:

Adresse hex	Adresse dez	x128	x64	x32	x16	x8	x4	x2	x1	Sum
F400	−3072	·	·	·	·	·	·	·	·	0
F401	−3071	·	·	·	·	·	·	·	·	0
F402	−3070	·	·	·	·	·	·	·	·	0
F403	−3069	·	·	·	·	·	·	·	·	0
F404	−3068	·	·	·	·	·	·	·	·	0
F405	−3067	·	·	·	·	·	·	·	·	0
F406	−3066	·	·	·	·	·	·	·	·	0
F407	−3065	·	·	·	·	·	·	·	·	0

Rechts neben der Tabelle unter dem Eintrag „Sum" stehen die Speicherinhalte (in diesem Fall Nullen). Mit

```
PRINT PEEK(-3072)
```

können wir dies leicht überprüfen.

Nun beginnen wir, auf dem quadratischen Feld das Zeichen X einzutragen und jeweils die Zeilensumme rechts zu vermerken:

Adresse hex	Adresse dez	x128	x64	x32	x16	x8	x4	x2	x1	Sum
F400	−3072	O	·	·	·	·	·	·	O	129
F401	−3071	·	O	·	·	·	·	O	·	66
F402	−3070	·	·	O	·	·	O	·	·	36
F403	−3069	·	·	·	O	O	·	·	·	24
F404	−3068	·	·	·	O	O	·	·	·	24
F405	−3067	·	·	O	·	·	O	·	·	36
F406	−3066	·	O	·	·	·	·	O	·	66
F407	−3065	O	·	·	·	·	·	·	O	129

Das Programmieren geschieht nun, indem die rechts ermittelten Summenwerte nacheinander in den Speicher für die frei programmierbaren Zeichen eingetragen werden:

```
POKE -3072,129
POKE -3071,66
POKE -3070,36
POKE -3069,24
POKE -3068,24
POKE -3067,36
POKE -3066,66
POKE -3065,129
```

Wenn Sie nun

```
PRINT CHR$(128)
```

aufrufen, erscheint ein frei definiertes X auf dem Bildschirm.

Dieses Verfahren ist recht mühsam, da Sie, um alle 128 Zeichen zu definieren, alle 1024 Adressen des frei programmierbaren Zeichenspeichers mit Werten versehen müssen.

In Abb. 2.17 finden Sie daher ein sogenanntes Hilfsprogramm, das Ihnen diese Arbeit erleichtert. Dieses Programm liest aus DATA-Zeilen, die vor dieses Hilfprogramm gesetzt werden und jeweils die Daten für einen kompletten Zeichensatz enthalten, die Werte ein, die in den programmierbaren Zeichenspeicher geladen werden sollen. Zusätzlich wird das programmierte Zeichen im Großformat auf dem Bildschirm dargestellt. Wie so etwas im Ablauf aussieht, sehen Sie in den Abbildungen 2.14 und 2.15.

Wir haben für dieses Programm mehrere komplette Zeichensätze zur Verfügung gestellt. Diese finden Sie in Form von DATA-Zeilen im Anhang C. Die Zeilennummer, die vor der DATA-Anweisung steht, entspricht dem ASCII-Code des jeweiligen Zeichens. Jeder Zeichensatz beginnt mit Programmzeile 33, wo ein neues ! definiert wird.

Sobald alle Zeichendefiniert sind, können Sie sich die Zeichen wahlweise im neuen oder im Standard-Zeichensatz ausgeben lassen (siehe Abb. 2.16).

Nun erfolgen Bildschirmausgaben im selbstdefinierten Zeichensatz. Wenn Sie dies nicht mehr wünschen, entfernen Sie Programmzeile 50 durch

```
50 <RETURN>
```

Alle Folgezeichen werden dann im Standardzeichensatz ausgegeben.

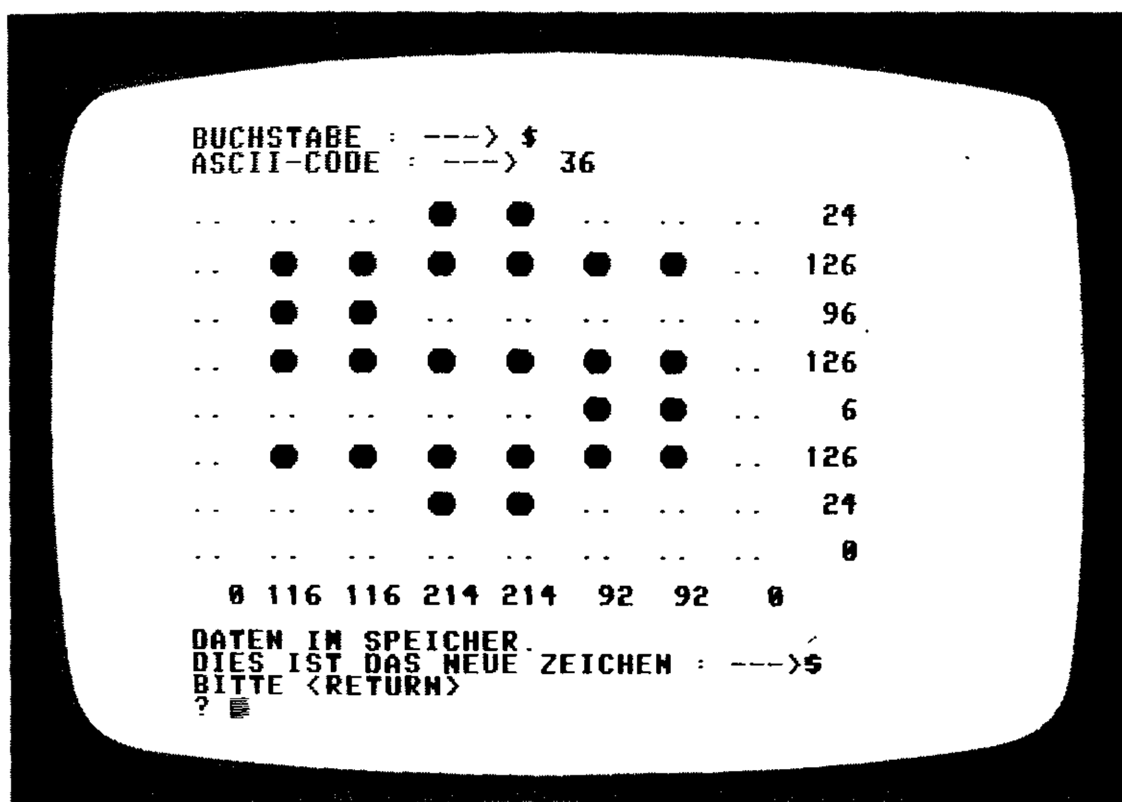


Abb. 2.14: Ein neues $-Zeichen wird definiert

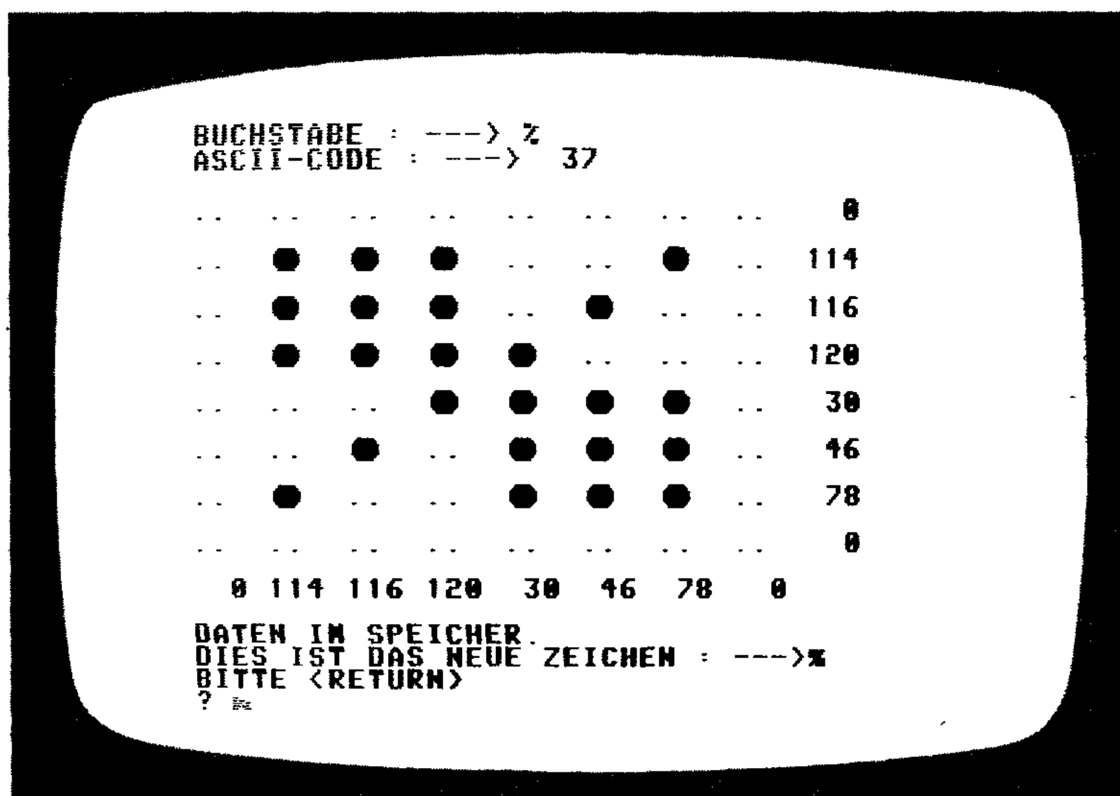


Abb. 2.15: Ein neues Prozentzeichen wird definiert

```
10 CLS:CHAR1
20 PRINTCHR$(14);
30 Y$=INKEY$:IFY$=""THEN30
40 A=ASC(Y$)
50 A=A+95
60 Y$=CHR$(A)
70 PRINTY$;
80 GOTO30
READY
>
```

Abb. 2.16: Wiedergabe eines beliebigen Zeichens

```
59999 'IN DIESEM PROGRAMM WERDEN 94 FREIDEFINIERBA
      RE ZEICHEN ALS BITMUSTER DEFINIERT UND IN DE
      N RECHNERSPEICHER GELADEN.FUER DIESEN TREIBE
      R STEHEN MEHRERE ZEICHENSAETZE ZUR VERFUEGUN
      G.
60000 CLEAR 5000:'PLATZ SCHAFFEN
60010 DEFINT M,B:'FUER BERECHNUNGEN ALS INTEGER NO
      TWENDIG
60020 DIM M(7,7):'DIESES FELD NIMMT DIE PUNKTE EIN
      ES ZEICHENS AUF
60030 DIM HW(7):'WELCHE BITS GESETZT
60040 DIM MW(7):'DATA WERTE
60050 ST=&HF400:'ABLAGEADRESSE
60060 FOR AN=0 TO 93
60070 CLS:PRINT"BUCHSTABE : ---> ";CHR$(33+AN)
60080 PRINT"ASCII-CODE : ---> ";33+AN
60090 PRINT
60100 FOR MX=0 TO 7:'ALLE 8 BYTES
60110 READ Z%
60120 HW(MX)=Z%
60130 FOR MB=0 TO 7:'ALLE 8 BITS EINES BYTES
60140 MM=2[MB:MW(MB)=(Z% AND MM):IF MW(MB)<>0 THEN
       MW(MB)=1
```

Abb. 2.17: Mit diesem Hilfsprogramm (in Verbindung mit einem der verfügbaren Zeichensätze aus Anhang C) wird ein ganzer Satz freiprogrammierbarer Zeichen in den Rechnerspeicher geladen

```
60150 M(MX,7-MB)=MW(MB):'IM FELD EINTRAGEN
60160 NEXT MB
60170 NEXT MX
60180 FOR Y=0 TO 7:'DARSTELLEN
60190 FOR X=0 TO 7
60200 IF M(X,Y)=0 THEN PRINT"..   ";ELSEPRINT"      "
      ;
60210 T%=2[(7-X):WW=WW+T%*M(X,Y)
60220 NEXT X
60230 PRINTUSING"###";WW
60235 PRINT
60240 POKE ST,WW:'IM SPEICHER FUER FREI DEFINIERBA
      RE ZEICHEN ABLEGEN
60250 ST=ST+1:WW=0
60260 NEXTY
60270 FORX=0TO7
60280 PRINTUSING"### ";HW(X);
60290 NEXTX
60300 PRINT:PRINT:PRINT"DATEN IM SPEICHER.
      DIES IST DAS NEUE ZEICHEN : --->";CHR$(128+A
      N)
60310 PRINT"BITTE <RETURN>"
60320 INPUT"";A$
60330 NEXT AN
```

Abb. 2.17: Mit diesem Hilfsprogramm (in Verbindung mit einem der verfügbaren Zeichensätze aus Anhang C) wird ein ganzer Satz freiprogrammierbarer Zeichen in den Rechnerspeicher geladen (Fortsetzung)

Diesem Listing müssen Sie einen der in Anhang C aufgeführten Zeichensätze vorausstellen.

2.3 FARBIGE DARSTELLUNG AUF DEM GRAFIKBILDSCHIRM

Schon auf den vorangegangenen Seiten wurde als wesentlicher Unterschied zwischen Textbildschirm und Grafikbildschirm herausgestellt, daß in der FGR-Bildschirmebene jeder Bildpunkt einzeln angesprochen werden kann. Hierbei stehen Ihnen 160 Bildpunkte pro Zeile und 102 pro Spalte zur Verfügung. Bevor Sie nun Bildpunkte darstellen können, müssen Sie sich 2 Farben wählen:

1. die Hintergrundfarbe;
2. die Farbe, in der Sie zeichnen wollen.

Die vier Ihnen insgesamt zur Verfügung stehenden Farben sind:

Farbnummer 1: schwarz
Farbnummer 2: blau
Farbnummer 3: orange
Farbnummer 4: grün

Wir wollen nun einen blauen Hintergrund definieren, auf dem wir einen Punkt in Orange zeichnen:

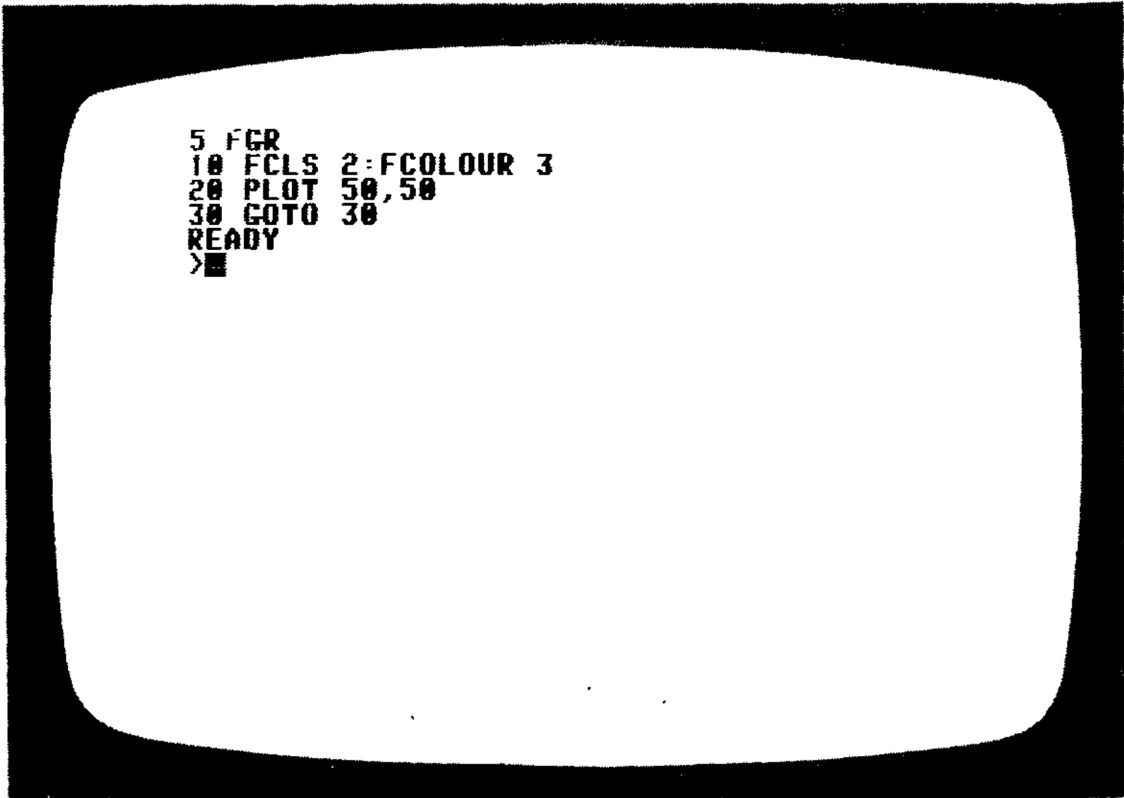


Abb. 2.18: Definition von Vorder- und Hintergrundfarbe

Es kann nun durchaus möglich sein, daß dieser Punkt auf Ihrem Farbfernseher in einer anderen Farbe oder etwas verschwommen erscheint. Sie sollten dann das folgende Programm eingeben. Es erzeugt ein 4-farbiges Testbild auf dem Bildschirm, mit dessen Hilfe Sie den UHF-Kanal Ihres Farbfernsehers nachjustieren können.

Eine entsprechende Bildschirmausgabe dieses Programms finden Sie in Abb. 2.20.

2.3.1 Zeichnen von Punkten, Linien und Kreisen

Zum Zeichnen von Punkten und Linien dient, wie Sie schon im vorherigen Kapitel erfahren haben, die PLOT-Anweisung. Zum Zeichnen von Punkten und Linien stehen Ihnen vier Farben zur Verfügung. Einen Punkt oder eine Linie löscht man, indem man als Zeichenfarbe dieselbe Farbe wie die Hintergrundfarbe verwendet. Ein einfaches Beispiel ist eine Punktewolke, bei der die Farbe und die Koordinaten des Punktes aus Zufallszahlen mit Hilfe der RND-Funktion ermittelt werden (Abb. 2.21).

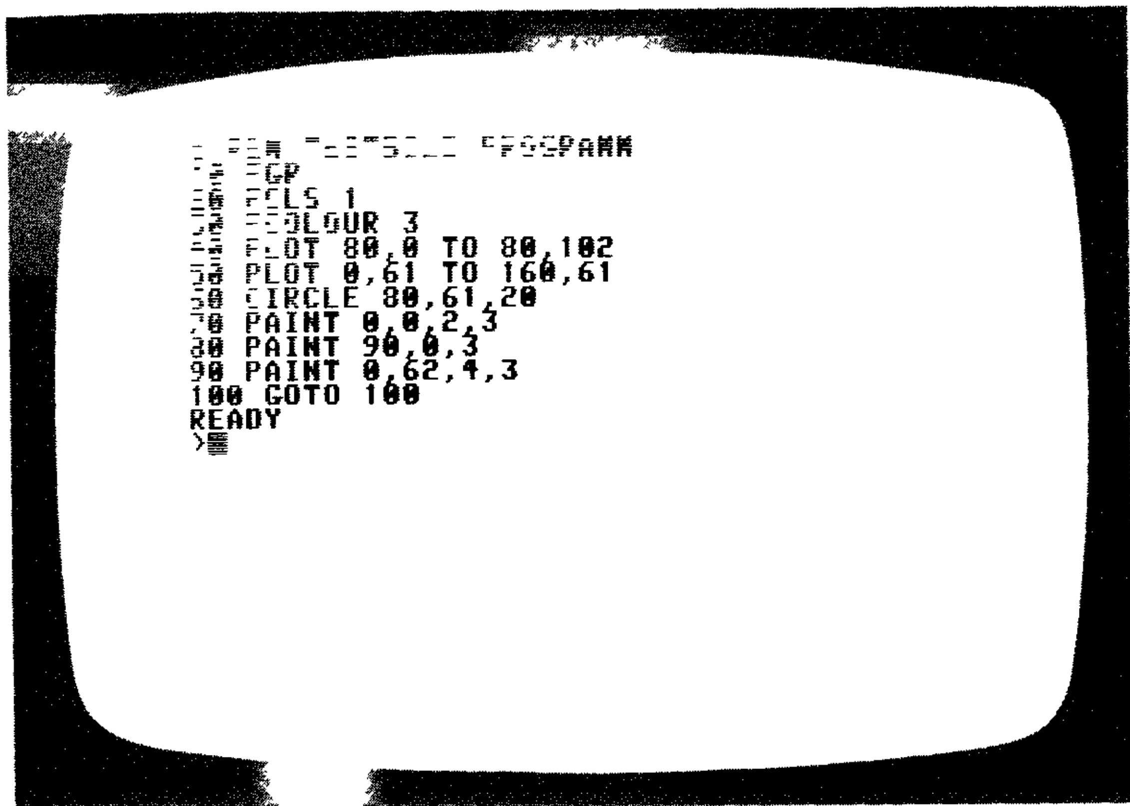


Abb. 2.19: Programm zur Erzeugung eines Testbilds

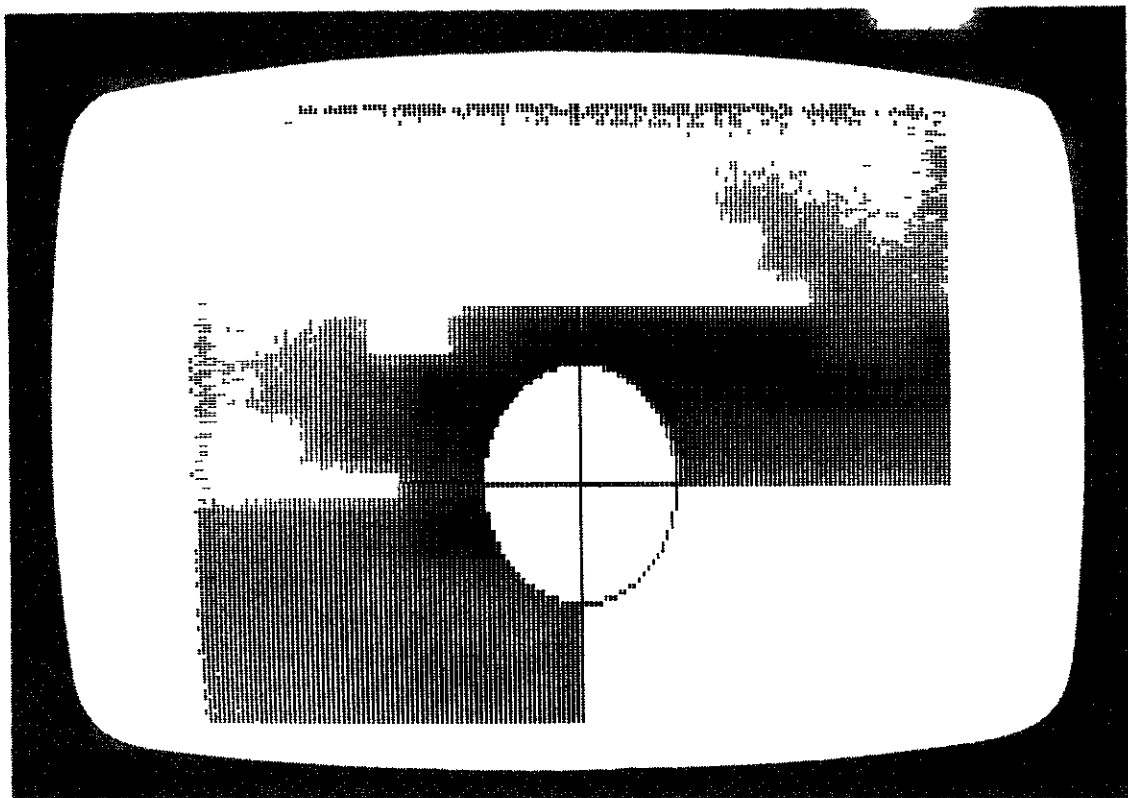


Abb. 2.20: Ein Testbild hilft bei der Justierung des Farbfernsehers.

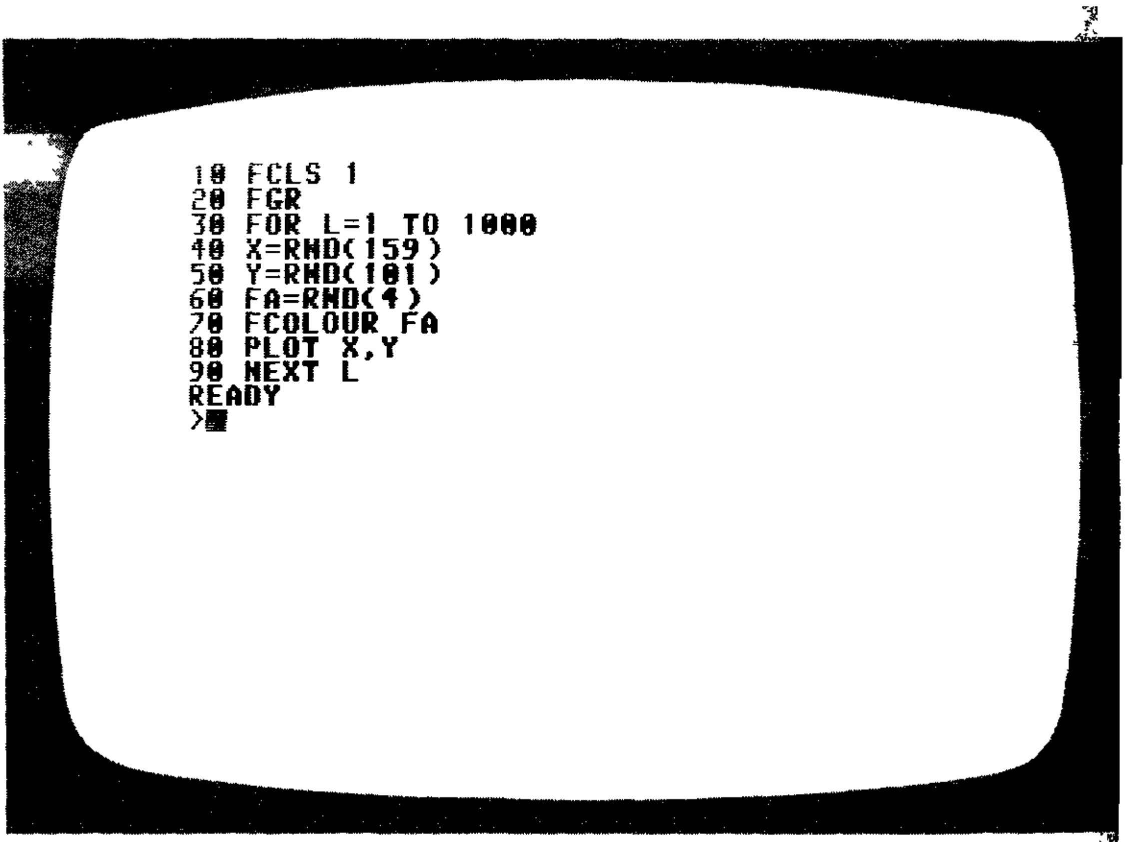


Abb. 2.21: Programm zur Erzeugung einer Punktewolke

Wird die PLOT-Anweisung in Programmzeile 80 mit einem TO oder einem zweiten Koordinatenpaar versehen, dann werden anstelle der Punkte Linien auf dem Bildschirm gezeichnet. Hier gibt es zwei Varianten, deren Unterschied Sie selbst nachvollziehen können, indem Sie im o. a. Programm folgende Änderungen vornehmen:

```
21 PLOT 0,0
80 PLOT TO X,Y
```

Bei dieser Variante wird immer eine neue Linie an den Endpunkt der vorherigen angehängt. Diese Form der PLOT-Anweisung ist sehr anwenderfreundlich ausgelegt, da die Anzahl der TO-Elemente nicht auf eines beschränkt ist:

```
PLOT 0,0 TO 159,0 TO 159,101 TO 0,101 TO 0,0
```

Durch diese Anweisung wird z. B. ein Rahmen um den FGR-Bildschirm erstellt. Dadurch werden gegenüber der anderen Variante eine Reihe von Befehlen und einiges an Arbeitsspeicher eingespart:

```
PLOT 0,0 TO 159,0:PLOT 159,0 TO 159,101:PLOT 159,101 TO
0,101:PLOT 0,101 TO 0,0
```

Diese Anweisungsfolge ist in ihrer Wirkung identisch mit der vorherigen.

Die Fähigkeit Ihres Colour-Genie, Linien zeichnen zu können, ist recht nützlich. Besonders interessant wird das Arbeiten mit dieser Grafik jedoch erst durch die Fähigkeit des Rechners, auch kreisförmige Darstellungen zu unterstützen. Dazu dient die CIRCLE-Anweisung, wie Sie sie in Programmzeile 50 sehen:

```
10 FCLS 1
20 FCOLOUR 3
30 FGR
40 FOR R=5 TO 50 STEP 5
50 CIRCLE RND(10)+70,RND(10)+38,R
60 NEXT R
70 FOR T=1 TO 500:NEXT T
80 GOTO 80
READY
>
```

Abb. 2.22: Erzeugung von Kreisen mit Hilfe der CIRCLE-Anweisung

Um einen Kreis zu zeichnen, müssen der CIRCLE-Anweisung drei Werte übergeben werden. Die ersten beiden kennzeichnen die X- und Y-Koordinate des Mittelpunkts, die dritte den Kreisradius.

Ganz rund erscheinen die Kreise eigentlich in den wenigsten Fällen. Alle Kreise sind ein wenig oval. Zwar läßt sich dies zum Teil vom Fernseher her korrigieren. Eine solche Korrektur wirkt sich jedoch auf die übrigen Bildschirmausgaben entsprechend aus und ist daher nicht empfehlenswert.

Umgangen werden kann dies, indem man die Bildpunktkoordinaten eines Kreises berechnet und einen Kreis über die PLOT-Anweisung darstellt. In diesem Falle können Sie nicht nur komplette Kreise, sondern auch Kreisbögen und Ellipsen konstruieren:

```
10    FCLS
20    FCOLOUR 3
30    INPUT"MITTELPUNKT WAAGERECHT";MX
40    INPUT"MITTELPUNKT SENKRECHT";MY
50    INPUT"KREISRADIUS";R
60    INPUT"ANFANGSWINKEL";AW
70    INPUT"ENDWINKEL";EW
80    IF EW=0 THEN EW=360
90    INPUT"X:Y VERHAELTNIS 1:";VY
100   IF VY=0 THEN VY=.9
110   FGR
120   PI=3.1415927
130   AW=AW/180*PI
140   EW=EW/180*PI
150   FOR A=AW TO EW STEP PI/(2*PI*R)
160   X=(R*COS(A)+MX)
170   Y=(R*SIN(A))
180   Y=VY*Y
190   Y=MY-Y
200   IF (X<0) OR (X>159) OR (Y<0) OR (Y>101) THEN
       230
210   IF A=AW THEN PLOT X,Y:GOTO230
220   PLOT TO X,Y
230   NEXT A
```

Abb. 2.23: Erzeugung eines Kreises durch Berechnung der einzelnen Koordinatenpunkte

Zeile 20 bis 90 machen den Eingabeteil des Programms aus. Zwingend vorgeschrieben sind hier die Eingaben für den Mittelpunkt und den Radius. Alle übrigen Eingaben können durch <RETURN> übergangen werden. In diesem Falle wird dann ein Vollkreis gezeichnet, bei dem die Länge des waagerechten gleich der des senkrechten Radius ist.

In Abb. 2.24 finden Sie eine Bildschirmausgabe dieses Programms, bei dem folgende Werte eingegeben wurden:

MITTELPUNKT WAAGERECHT ? 80
MITTELPUNKT SENKRECHT ? 50
KREISRADIUS ? 50
ANFANGSWINKEL ? 0
ENDWINKEL ? 360
X–Y VERHÄLTNIS 1:1

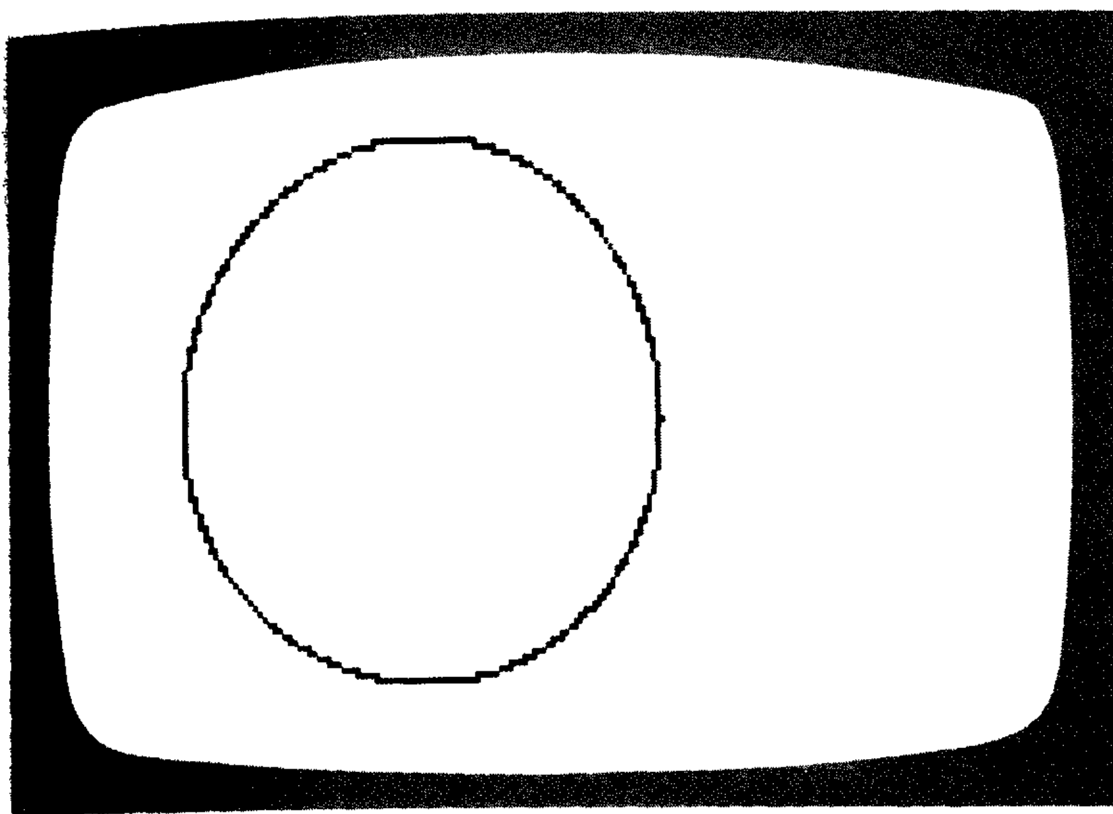


Abb. 2.24: Ein mit dem Programm aus Abb. 2.23 erstellter Kreis

2.3.2 Farbiges Ausfüllen von Flächen

Eine leistungsstarke Anweisung des Colour-Genie ist der PAINT-Befehl. Mit dieser Hilfe können Sie beliebige Flächen, die durch die Kantenlinien einer Figur vorgegeben sind, farbig ausmalen. Starten Sie dazu noch einmal das Programm aus Abb. 2.23 (Ellipse/Kreis) mit den dort vorgegebenen Werten. Wie Sie den Programmzeilen 10 und 20 entnehmen können, wurde in diesem Programm die Farbe 1 als Hintergrundfarbe und die Farbe 3 als Zeichenfarbe gewählt. Diese Information ist wichtig, da durch die PAINT-Anweisung immer so lange Flächen farbig ausgemalt werden, bis der Rechner auf dem Grafikbildschirm eine bestimmte Farbe vorfindet, die er als Begrenzungsfarbe auffaßt. Die Farbe Orange (Farbe 3) ist also auf die Kreisfläche bezogen eine Begrenzungsfarbe. Um dieses Kreis mit grüner Farbe auszumalen, geben Sie einfach folgende Anweisung ein:

```
PAINT 80,50,4,3
```

Diese Anweisungsfolge bedeutet für den Rechner folgendes:

Male so lange die Kreisfläche, beginnend mit der Bildschirmkoordinate 80,50 in der Farbe Grün (Nr. 4) aus, bis du an orangefarbene (Farbe 3) Grenzen stößt. Dann beende den Zeichenvorgang.

Zur PAINT-Anweisung gehört also ein Koordinatenpaar, eine Zeichenfarbe und eine Begrenzungsfarbe.

Nun kann es vorkommen, daß die Linien, die eine farbig auszufüllende Fläche begrenzen, selbst verschiedene Farben besitzen, wie in folgendem Beispiel:

```
10 FCLS 1
20 FCOLOUR 2
30 FGR
40 PLOT 1,10 TO 100,4
50 FCOLOUR 3
60 PLOT TO 60,60 TO 1,10
70 GOTO 70
```

Abb. 2.25: Eine Dreiecksfläche soll farbig ausgemalt werden

Diese Dreieckseiten sind mehrfarbig. Um diese Fläche farbig ausfüllen zu können, müssen sowohl Farbe 2 als auch Farbe 3 als Begrenzungsfarbe ausgewiesen werden. Soll wiederum grün ausgemalt werden (Farbe 4), so lautet die entsprechende Anweisung:

```
PAINT 60,20,4,2,3
```

2.4 DAS ARBEITEN MIT BILDSCHIRMSTEUERZEICHEN

Obwohl Sie vielleicht nicht so sehr darauf geachtet haben, kennen Sie schon eine Reihe von Bildschirmsteuerzeichen aus dem Kapitel „Ein Grafikeditor“. In der dort gezeigten Tastatureingaberoutine wurden den Pfeiltasten auf der Tastatur neue Funktionen zugewiesen. Hier noch einmal dieser Programmbestandteil:

```
10 CLS
20 COLOUR 1
30 PRINT CHR$(14);
40 Y$=INKEY$:IFY$=""THEN40
50 IFY$=CHR$(13)THENY$=CHR$(29)+CHR$(26)
60 IFY$=CHR$(91)THENY$=CHR$(27)
70 IFY$=CHR$(10)THENY$=CHR$(26)
80 IFY$=CHR$(9)THENY$=CHR$(25)
90 IFY$=CHR$(8)THENY$=CHR$(24)
100 PRINTY$;:GOTO40
```

Abb. 2.26: Tastatureingaberoutine

Alle aufgeführten ASCII-Codes in diesem Programm, deren Werte kleiner sind als 32, repräsentieren sog. Steuerzeichen. Sie tragen diesen Namen deshalb, weil sie nicht, wie alle übrigen ASCII-Codes, ein auf dem Bildschirm darstellbares Zeichen repräsentieren, sondern

a) entweder die Position oder Darstellung des Cursors auf dem Bildschirm verändern oder
b) den Bildschirm oder Teile davon löschen.

Wenn Sie in Ihrem Colour-Genie-Handbuch die Seite 145 aufschlagen, finden Sie dort eine sog. SPECIAL CODE Tabelle. Dort werden die Funktionen einiger Bildschirmsteuerzeichen erläutert. Nun wissen Sie z. B., daß Zeile 30 des o. a. Programms den Cursor einschaltet. Auch die Funktion der übrigen Bildschirmsteuerzeichen können Sie dieser Tabelle entnehmen.

Sie sollen an einem Beispiel erkennen, wie man Steuerzeichen effektiv einsetzen kann:

```
10    CLS:CLEAR2000
20    PRINT@40,"DIESER TEXT BLEIBT UNBEEINFLUSST";
30    A$="MELDUNG 1 IST EIN ETWAS LAENGERER TEXT"
40    B$="MELDUNG 2 IST KUERZER"
50    PRINT@0,;
60    PRINTA$;
70    FOR T=1 TO 500:NEXT T
80    PRINT@0,;
90    PRINTB$;
100   GOTO 100
```

Abb. 2.27: Ein Beispiel für die Anwendung von Bildschirmsteuerzeichen

Wenn Sie dieses Programm starten, wird zuerst links oben auf dem Bildschirm MELDUNG 1 IST EIN ETWAS LAENGERER TEXT ausgegeben. Nach einigen Sekunden wird dieser Text durch die neue Information MELDUNG 2 IST KUERZER überschrieben. Trotzdem bleibt der hintere Teil des vorher ausgegebenen Textes nach wie vor auf dem Bildschirm stehen. Diesen unerwünschten Begleiteffekt können Sie einfach beseitigen, indem Sie ein Steuerzeichen verwenden, das alle Texte von der Cursorposition bis zum Zeilenende löscht. Dieses Zeichen hat den ASCII-Code 30.

```
85 PRINT CHR$(30);
```

Starten Sie mit dieser Programmzeile das Programm neu. Nun finden keine Überlappungen der ausgebenen Texte mehr statt.

Nur wenige Steuerzeichen sind direkt über die Tastatur ansprechbar. Wenn Sie sich Ihre Tastatur anschauen, fallen Ihnen sicherlich zuerst die

Pfeiltasten ins Auge. Wie können Sie nun feststellen, welcher ASCII-Code bei Betätigung einer Taste von der Tastatur an den Rechner weitergegeben wird?

Lassen Sie uns systematisch vorgehen. Sie betätigen eine Taste, und im Rechner findet eine Reaktion darauf statt. Das Colour-Genie hat also von der Tastatur ein Zeichen eingelesen und weiterverarbeitet. Bevor wir wissen, worin diese Weiterverarbeitung bestand, müssen wir dieses Zeichen kennenlernen. Benötigt wird also ein Programm, das eine Tastaturabfrage durchführt und, sobald eine Tastenbetätigung erfolgt, das betreffende Zeichen erfaßt und dessen Charakteristika auf dem Bildschirm ausgibt. Da alle Zeichen, über die Ihr Colour-Genie verfügt, in einer sog. ASCII-Tabelle registriert und mit einem Wert von 1 bis 255 belegt sind, können Sie mit Hilfe der BASIC-Anweisung PRINT ASC (betätigte Taste) sich diesen Wert ausgeben lassen und die gewünschten Rückschlüsse auf Art und Funktion des Zeichens ziehen.

Sie benötigen also erst einmal ein Programm, das die Tastaturabfrage durchführt und in der Lage ist, ein Zeichen in einer Variablen, z. B. X$, zu erfassen:

```
10 X$=INKEY$
20 IF X$=""THEN 10
30 PRINT"Die betaetigte Taste liefert das
Zeichen mit dem ASCII-Code :";ASC(X$)
40 GOTO 10
```

Abb. 2.28: Der ASCII-Code einer betätigten Taste wird ausgegeben

Starten Sie das Programm, und betätigen Sie die Taste A. Es erscheint folgendes auf dem Bildschirm:

Die betaetigte Taste liefert das Zeichen mit dem
ASCII-CODE: 65

Wenn Sie nun in Ihrer ASCII-Code-Tabelle auf Seite 145 nachschauen, werden Sie dies bestätigt finden. Sie können auch andere Tasten, wie 1, (usw. betätigen und entsprechende Vergleiche in der Tabelle durchführen.

Betätigen Sie nun einmal die Taste <Pfeil links>. Sie sehen folgende Bildschirmausgabe:

Die betaetigte Taste liefert das Zeichen mit dem
ASCII-CODE: 8

Wenn Sie nun wieder auf die ASCII-Tabelle zurückgreifen, finden Sie dort in der ersten Spalte hinter der 8 den Text BS. Das ist die Abkürzung

für den Namen dieses Steuerzeichens: „Backspace“, das heißt „rückwärts“. Im Tabellenteil SPECIAL-Codes finden Sie auch eine Erläuterung dazu: „Löscht letztes Zeichen“.

Nehmen Sie sich nun bitte ein Blatt Papier zur Hand, und notieren Sie sich die ASCII-Zeichen für

Pfeil links	<8>
Pfeil rechts	<9>
Pfeil runter	<10>
Pfeil hoch	<91>
RETURN	<13>
SHIFT Pfeil links	<24>
SHIFT Pfeil rechts	<25>
SHIFT Pfeil runter	<26>
SHIFT Pfeil hoch	<27>
SHIFT RETURN	<13>

Die Ergebnisse müßten mit den Werten in Klammern identisch sein. Ändern Sie nun bitte unser Programm wie folgt:

```
5 CLS
30 PRINT X$;
```

Starten Sie es, geben Sie beliebigen Text ein, und experimentieren Sie mit den Pfeiltasten und deren <SHIFT>-Kombinationen. Vergleichen Sie jedesmal anhand Ihrer Mitschrift, welches Steuerzeichen auf dem Bildschirm ausgegeben wurde, welche Funktion es laut ASCII- und SPECIAL-CODE-Tabelle besitzt und wie seine Auswirkungen auf den Bildschirmtext sind. Ihnen wird auffallen, daß die Betätigung der RETURN-Taste z. B. den unangenehmen Nebeneffekt besitzt, zwar auf die nächste Zeile zu verzweigen, dort eventuell stehenden Text jedoch zu löschen. Lassen Sie uns diesen Nebeneffekt beseitigen.

In der SPECIAL-Code-Tabelle finden Sie die Steuerzeichen für „Cursor zum Zeilenanfang“ und „Cursor eine Zeile tiefer“ als CHR$(29) und CHR$(26) definiert. Ebenfalls wissen Sie anhand Ihrer Aufzeichnungen, daß die RETURN-Taste den ASCII-Code 13 liefert und dieser in der Variablen X$ als CHR$(13) geführt wird.

Wenn Sie nun eine Zeile 25 definieren,

```
25 IF X$=CHR$(13) THEN X$=CHR$(29)+CHR$(26)
```

ist der „Nebeneffekt“ weg, und Sie können auf einer ganzen Bildschirmseite Text erstellen und korrigieren.

Kapitel 3

Einsatz als Tischrechner

Dieses und die folgenden Kapitel sollen Ihnen helfen, sich mit dem Colour-Genie und seinen Eigentümlichkeiten vertraut zu machen. Dazu gehört eine Menge Theorie, die solche Informationen beinhalten soll, denen Sie, soweit sie im Handbuch aufgeführt waren, bestimmt nicht viel Beachtung schenken.

Das folgende Kapitel ist den mathematischen Grundlagen gewidmet.

3.1 ZAHLENDARSTELLUNG

Der Rechner unterscheidet zwischen ganzen Zahlen und Gleitkommazahlen. Für ganze Zahlen werden im Rechner zwei Speicherplätze belegt. Ein solcher Speicherplatz wird Byte genannt. Da jedes Byte Zahlenwerte zwischen 0 und 255 erfassen kann, können folglich mit zwei Bytes 256 mal 256, also 65536 verschiedene Zahlen dargestellt werden. Leider braucht man zum Arbeiten mit dem Rechner auch negative ganze Zahlen. Aus diesem Grunde wurde festgelegt, daß mit zwei Bytes der Zahlenbereich −32767 bis 32767 dargestellt werden soll. Sollte Ihnen noch unklar sein, warum dies so ist, dann lesen Sie bitte nochmal in Ihrem Handbuch Kapitel 32 Seite 104ff. die Erläuterungen zu Bit, Byte und Hexadezimalzahlen nach. Dort wird auf den Seiten 105 und 106 beschrieben, warum man die hexadezimale Zahl F000 unterschiedlich interpretieren kann, nämlich einmal als negative Zahl −4096 und zum anderen als positive Zahl 61440.

In der Praxis wird häufig dann mit ganzen Zahlen gerechnet, wenn Speicheradressen berechnet und Zählungen vorgenommen werden oder wenn aus sonstigen Gründen unbedingt vermieden werden muß, daß bei Berechnungen Nachkommastellen auftreten. Soll der Rechner nur mit ganzzahligen Werten arbeiten, muß ihm das explizit über den Variablennamen mitgeteilt werden. Dies geschieht durch Anhängen des Prozentzeichens an den Variablennamen oder alternativ durch die DEFINT-Anweisung (s. a. BASIC-Handbuch Seite 13 und 72).

Verfolgen Sie in eigenen Programmen eine solche Linie konsequent, indem Sie bestimmte Variablen von vornherein als ganzzahlig definieren, dann können Sie eine Menge Speicherplatz sparen.

Da ansonsten im Microsoft-BASIC mit Gleitkommazahlen gearbeitet wird, werden auch ganze Zahlen in Gleitkommazahlen umgewandelt und weiterverarbeitet. Nur wenn explizit per Definition durch die DEFINT-Anweisung oder durch die Variablenkennung % eine Zahl als ganzzahlig festgelegt ist, wird diese auch als solche gespeichert.

Im Microsoft-BASIC kann mit zwei verschiedenen Arten von Gleitkommazahlen gerechnet werden. Dies verdeutlicht folgendes Beispiel:

```
PRINT 22/7
3.14286
READY
>
```

einerseits und

```
PRINT 22#/7#
3.142857142857143
READY
>
```

andererseits.

Im ersten Fall ist das Ergebnis sechs-, im zweiten Fall sechzehnstellig. Man spricht hier von Gleitkommazahlen einfacher und doppelter Genauigkeit. Was ist überhaupt eine Gleitkommazahl?

Zur Erzeugung einer Gleitkommazahl wird jeder eingegebene Zahlenwert zunächst einmal in eine andere Schreibweise umgerechnet. Man nennt diesen Vorgang Normieren. Hierbei ermittelt der Rechner eine Zahl zwischen 0.5 und 1, die, multipliziert mit einer Potenz von 2, dem Wert der eingegebenen Zahl entspricht, z. B.:

$6 = 0.75 * 2^3$

Hier wurde beim Normieren eine Zahl zwischen 0.5 und 1 ermittelt, nämlich 0.75. Für diese Mantisse stehen drei Bytes zur Verfügung, wenn es sich um eine Gleitkommazahl mit einfacher Genauigkeit handelt, und sieben Bytes, wenn es sich um eine Gleitkommazahl mit doppelter Genauigkeit handelt. Die Umrechnung einer Zahl in eine Mantisse und einen Exponenten geht nicht immer auf. Es ergeben sich Fehler, deren Ursache darin zu finden ist, daß eben nur für eine gewisse Anzahl Nachkommastellen Platz in den für eine Zahl zur Verfügung gestellten Bytes ist. Will man z. B. den Wert 7.9999999999 in der Variablen A# speichern und nicht den Wert 8, ist es notwendig, auf Gleitkommazahlen doppelter Genauigkeit auszuweichen. Dies können Sie leicht nachvollziehen:

```
X=7.9999999999
PRINT X
8
READY
>
```

und

```
A#=7.9999999999
PRINT A#
7.9999999999
READY
>
```

Der Sinn dieser Normierung besteht darin, in möglichst wenigen Bytes möglichst viele signifikante Ziffern zu erfassen. Dies wird leicht verständlich, wenn man die Zahlen 0.12345678 mal 10 hoch −6 und 0.00000012 betrachtet. Beide Zahlen würden zur Erfassung die gleiche Anzahl von Bytes benötigen.

Das Microsoft-BASIC verfügt übrigens über eine Funktion, die VARPTR (Variable) heißt und eine wertvolle Hilfe darstellt, wenn es darum geht, rechnerinterne Vorgänge wie die Normierung von Gleitkommazahlen nachzuvollziehen. Dieser VARPTR-Befehl ermittelt für eine Variable genau die Adresse im Speicher, die für eine Wertzuweisung dieser Variablen reserviert wurde. Wenn Sie also irgendwo in einem Programm eine Variable als

```
A#=6
```

definiert haben und nun wissen wollen, wo die Zahl 6 als Gleitkommazahl doppelter Genauigkeit abgelegt wurde, kann Ihnen diese VARPTR-Funktion eine wertvolle Hilfe sein. Dazu folgendes Beispiel:

```
10 A#=6
20 PRINT VARPTR(A#)
```

Wenn Sie dieses Programm starten, gibt Ihnen der Rechner eine Adresse innerhalb seines Programmspeichers aus, wo er für die Variable A mit dem Wert 6 Platz geschaffen hat.

Mit Hilfe des folgenden Programms können Sie den Rechner anweisen, Ihnen weitere Informationen über Mantisse und Exponent zu erteilen:

```
10 CLS:PRINT"SO WIRD DER WERT FUER DIE V
ARIABLE A# INSPEICHER ABGELEGT:"
20 A#=6:'A# IST DIE DEF. VARIABLE
30 AD=VARPTR(A#):'AD=ERMITTELTE ADRESSE
40 FOR X=AD TO AD+7
50 PRINT"WERT ";X-AD+1;": ";
60 W=PEEK(X)
70 PRINTUSING"###";W;:PRINT" DEZIMAL = "
;
80 PRINTMID$("0123456789ABCDEF",INT(W/16
)+1,1);MID$("0123456789ABCDEF",W-INT(W/1
6)*16+1,1);" HEXADEZIMAL."
90 'DIE IN ZEILE 80 VERWENDETE FUNKTION
WIRD EINGEHEND IN KAPITEL 3.7 ERLAEUTERT

100 NEXT X
READY
>
```

Abb. 3.1: Der Interpreter kann die Adressen ermitteln, in denen er die Werte ablegt, die Variablen zugewiesen wurden.

```
SO WIRD DER WERT FUER DIE VARIABLE A# IM
SPEICHER ABGELEGT:
WERT  1 :    0 DEZIMAL = 00 HEXADEZIMAL.
WERT  2 :    0 DEZIMAL = 00 HEXADEZIMAL.
WERT  3 :    0 DEZIMAL = 00 HEXADEZIMAL.
WERT  4 :    0 DEZIMAL = 00 HEXADEZIMAL.
WERT  5 :    0 DEZIMAL = 00 HEXADEZIMAL.
WERT  6 :    0 DEZIMAL = 00 HEXADEZIMAL.
WERT  7 :   64 DEZIMAL = 40 HEXADEZIMAL.
WERT  8 :  131 DEZIMAL = 83 HEXADEZIMAL.
READY
>
```

Abb. 3.2: So wird die Gleitkommazahl 6 im Speicher abgelegt

Der Rechner bringt eine Bildschirmausgabe wie in Abb. 3.2.

Was sagt diese Bildschirmdarstellung aus?

Die Zahl 6, so wurde weiter oben festgestellt, entspricht normiert dem Wert 0.75 * 2 hoch 3. Die rechnerinterne Mantissendarstellung für 0.75 ist

0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Das erste Bit, in diesem Fall 0, repräsentiert das Vorzeichen (es handelt sich also um eine positive Zahl). Das nächste Bit, das einzige gesetzte in dieser Mantisse, repräsentiert den Wert 0.25. Das danach folgende würde den Wert 0.125 repräsentieren. Es fällt auf, daß das Bit für den Wert 0.5 fehlt, denn 0.5 plus 0.25 ergibt erst 0.75. Dieses Bit wird gespart, da man ja generell weiß, daß die Zahlen ab 0.5 aufwärts liegen und folglich immer 0.5 in ihnen enthalten ist.

Der Sinn dieser Normierung ist darin zu sehen, möglichst viele signifikante Ziffern zu speichern. Die Zahl 0.123456789 * 10 hoch −8 enthält verständlicherweise mehr Informationen als die Zahl 0.0000000001. Würden also z. B. bei sehr kleinen Zahlenwerten führende Nullen auch als solche abgespeichert, ginge eine Menge Speicherplatz verloren. Daß dem keinesfalls so ist, können Sie dem Beispielprogramm in Abb. 3.1 entnehmen, indem Sie in Zeile 20 für die Variable A# andere Zahlen, z. B. solche mit extrem vielen Nachkommastellen, einsetzen. Solange die Anzahl der signifikanten Ziffern sechzehn nicht überschreitet, wird die Gleitkommazahl korrekt in acht Bytes abgespeichert.

Neben der Mantisse wird bei der Normierung auch der Exponent ermittelt. Für die Zahl 6 ist dies der Wert 3. Daß wir in Abb. 3.2 unter dem Wert 8 dennoch keine 3, sondern die Zahl 131 vorfinden, beruht auf rechentechnischen Gründen. Bei der Normierung wird zu jedem ermittelten Exponenten die Zahl 128 hinzuaddiert. Jede im Microsoft-BASIC ermittelte Gleitkommazahl benötigt für die Erfassung des Exponenten ein Byte. Eine Gleitkommazahl benötigt also insgesamt 4 Bytes, wenn sie in einer Variablen einfacher Genauigkeit erfaßt wurde (mit maximal sechs signifikanten Ziffern), und 8 Bytes, wenn sie in einer Variablen dopelter Genauigkeit erfaßt wurde; hier können sechzehn signifikante Ziffern erfaßt werden.

3.1.1 Exponentielle Schreibweise

Neben der allgemeinen Schreibweise für Gleitkommazahlen, wie die Eingaben 3.1415927 oder −99.53, gibt es die Schreibweise in Exponentialform. Hier schreibt man

4.318E4

und meint nichts anderes als 4.318 * 10 hoch 4.

In diesem Format steht also rechts vom E die Anzahl der Stellen, um die das Komma in der Mantisse nach rechts, oder z. B. bei der Zahl 4.318E−1, nach links gerückt werden muß. Bei der Exponentialschreibweise im Microsoft-BASIC wird anstelle des Buchstabens E oft der Buchstabe D verwendet. Dadurch wird dem Rechner mitgeteilt, daß die eingegebene Zahl als Gleitkommazahl doppelter Genauigkeit behandelt werden soll. Hierbei ist zu beachten, daß z. B. bei einer Eingabe der Rechner nur dann mit doppelter Genauigkeit rechnet bzw. rechnen kann, wenn die Variable, der dieser Wert zugewiesen wird, eine Variable doppelter Genauigkeit ist. Zwar ist auch für eine Variable einfacher Genauigkeit eine Eingabe der Form

0.1234567890123D−1

möglich, diese Zahl wird aber dann als .0123457 gespeichert.

Die Eingabe von Gleitkommazahlen mit doppelter Genauigkeit ist also nur dann angebracht, wenn diese auch vom Rechner in doppelter Genauigkeit weiterbearbeitet werden kann.

In diesem Zusammenhang ist noch ein weiterer Hinweis von Bedeutung:

Wird eine Eingabe in Exponentialschreibweise vorgenommen, ist es dringend erforderlich, daß der Buchstabe E oder D, wie in den oben aufgeführten Beispielen, als Groß- und nicht als Kleinbuchstabe eingegeben wird. Sollte sich trotzdem ein kleiner Buchstabe in einer Eingabe verirren, wird diese vom Rechner ignoriert, die Meldung

```
?REDO
```

ausgegeben und eine neue Eingabe verlangt. Aus der Anzahl der zur Verfügung stehenden Bytes zur Erfassung einer Gleitkommazahl einfacher bzw. doppelter Genauigkeit läßt sich auch die kleinste bzw. größte erfaßbare Zahl ermitteln: Als zulässiger Zahlenbereich gilt −1E38 bis +1E38, genauer gesagt:

−1.701411E+−38 bis +1.701411E+−38

für Gleitkommazahlen einfacher Genauigkeit und

−1.70141183454455D+−38 bis +1.70141183454455D+38

für Variablen doppelter Genauigkeit.

Auch wenn Sie sich vom Rechner Zahlen ausgeben lassen, werden Sie manchmal feststellen, daß deren Werte, sobald sie über bzw. unter einer bestimmten Grenze liegen, in der exponentiellen Schreibweise ausgegeben werden. Dies tritt in der Regel dann ein, wenn die Zahlen betragsmäßig außerhalb des Bereichs von 0.01 bis 999999.4 liegen.

Mit der Ausgabe von Gleitkommazahlen machen Sie sich am besten vertraut, indem Sie folgendes kurze Programm eingeben und selbst experimentieren:

```
10 INPUT A#
20 PRINT A#
30 GOTO 10
```

Starten Sie dieses Programm, und geben Sie Zahlenwerte wie 24E5, −3.6E−4, 8E6 und andere Gleitkommazahlen Ihrer Wahl ein. Sie sollten erst dann zum nächsten Kapitel übergehen, wenn Sie sicher sind, diese Thematik verstanden zu haben.

3.1.2 Zahlenausgabe mit PRINT USING

Es kann auch Situationen geben, bei denen es wünschenswert ist, Zahlen, die wegen Ihrer Größe normalerweise in Exponentialschreibweise ausgegeben würden, in „normaler Schreibweise" ausgeben zu lassen. In einer Tabelle macht es sich z. B. nicht besonders gut, wenn dort eine Zahl wie 6.45E8 auftaucht. In diesem Fall können Sie sich einer BASIC-Anweisung bedienen, die PRINT USING heißt.

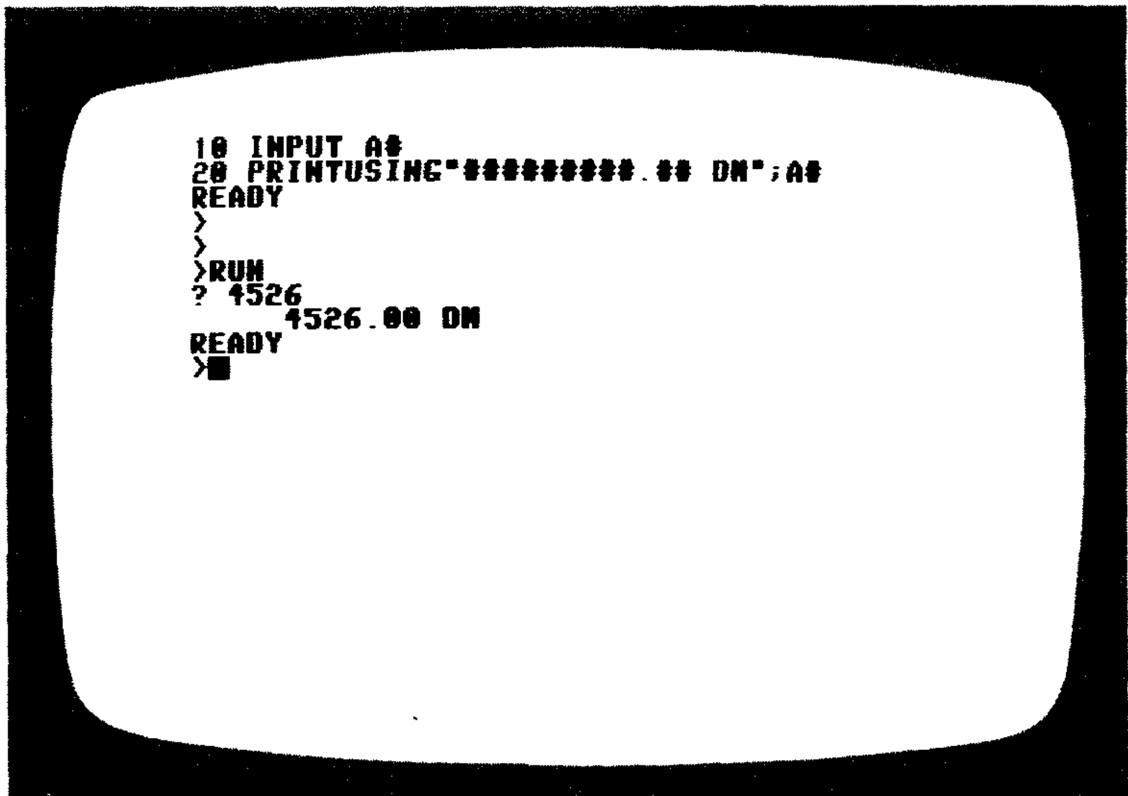


Abb. 3.3: Mit Hilfe der PRINT USING-Anweisung können Sie Ausgabeformate vorgeben

Mit ihr können Sie ein gewünschtes Ausgabeformat selbst vorgeben. Diese Anweisung finden Sie eingehend auf den Seiten 64ff. Ihres Handbuchs beschrieben. Aus diesem Grunde soll hier auch nur ein kurzes Beispiel vorgestellt und nicht weiter erläutert werden.

Angenommen, Sie rechnen im obigen Beispiel nur mit DM-Beträgen, d. h., daß auszugebene Beträge maximal zwei Nachkommastellen und den Zusatz DM aufweisen sollen. Dazu müssen Sie Zeile 20 wie folgt abändern:

```
20 PRINT USING"#########.## DM";A$
```

Eine entsprechende Bildschirmausgabe finden Sie in Abb. 3.3.

3.2 OPERATIONEN

Die üblichen Regeln der Punkt- und Strichrechnung sind Ihnen sicherlich vom Taschenrechner her bekannt. Sie sollen hier aus diesem Grunde nicht detailliert behandelt werden, da Computer mit arithmetischen Operationen nicht anders verfahren als Taschenrechner. Auch hier gelten die Regeln:

Klammerrechnung geht vor Punktrechnung.
Punktrechnung geht vor Strichrechnung.

Ihr Colour-Genie-Handbuch widmet das ganze Kapitel 3 diesem Sachverhalt.

Was Ihnen allerdings nicht so geläufig sein wird wie arithmetische Operationen sind logische Operationen. Das Microsoft-BASIC kennt die drei logischen Operatoren

AND OR und NOT.

Das Ergebnis einer solchen logischen Operation ist entweder der Wert Null oder ein Wert ungleich 0, in der Regel −1. Der Zahlenwert Null wird hierbei vom Rechner als „falsch" interpretiert, der Wert −1 als „wahr".

Mit logischen Operationen prüft man demzufolge den Wahrheitsgehalt von Aussagen. Wenn Sie den Rechner Bedingungen prüfen lassen, z. B. in Form einer IF-Anweisung, fällt Ihnen dies in der Regel gar nicht auf,

```
IF A=5 OR A=3 THEN . . .
```

da hier das Ergebnis einer logischen Operation gar nicht offen zu Tage tritt, sondern sofort vom Rechner intern weiterverarbeitet wird. Daß jedoch auch Aussagen wie

```
PRINT 1 AND 5
```

oder

```
PRINT 0 OR 1
```

logische Operationen darstellen und auch in dieser Weise vom Rechner verarbeitet werden, ist der Grund dafür, daß wir uns näher mit diesen Dingen befassen müssen. Zunächst sollen Sie die Wahrheitstafeln der Operatoren AND, OR und NOT kennenlernen.

OR	0	−1
0	0	−1
−1	−1	−1

AND	0	−1
0	0	0
−1	0	−1

NOT	0	−1
	−1	0

Dies sind also sogenannte Wahrheitstafeln für Operationen. Solche Wahrheitstafeln geben immer Auskunft darüber, wie das Ergebnis einer logischen Operation ist, wenn

a) beide der geprüften Bedinungen falsch sind;
b) entweder die eine oder die andere Bedingung wahr ist;
c) beide geprüften Bedingungen wahr sind.

Stellen Sie sich einmal vor, sie wollen den Wahrheitsgehalt der Aussage

Ich heiße Frank UND bin im Dezember geboren

prüfen, so kann diese Aussage entweder wahr oder falsch sein, je nachdem, welche der folgenden Bedingungen als wahr oder falsch gegeben sind:

1. Ich heiße nicht Frank (0) UND ich bin im November geboren (0)
2. Ich heiße Frank (−1) UND bin im November geboren (0)
3. Ich heiße Willi (0) UND bin im Dezember geboren (−1)
4. Ich heiße Frank (−1) UND bin im Dezember geboren (−1)

Nur die letzte geprüfte Bedingung, Nr. 4, liefert in ihrer Gesamtaussage ein logisches „Wahr". Dies sagt auch die Wahrheitstafel aus:

0	AND	0	ergibt	0	(Fall 1)
−1	AND	0	ergibt	0	(Fall 2)
0	AND	−1	ergibt	0	(Fall 3)
−1	AND	−1	ergibt	−1	(Fall 4)

Trotz alledem wird aus solchen Beispielen der Wahrheitsgehalt einer Aussage nicht immer ersichtlich. So ist z. B. die Aussage

Ich gehe heute schwimmen ODER surfen

in der Beziehung zweideutig, da sie entweder bedeuten kann:

Fall 1 Ich gehe heute entweder	schwimmen ODER ABER surfen
Fall 2 Ich gehe heute schwimmen,	(ODER) surfen ODER vielleicht auch beides.

Spricht man vom „Schwimmen oder Surfen" und meint den ersten Fall, so ist der Wahrheitsgehalt der Aussage als falsch zu betrachten, wenn sowohl das eine als auch das andere getan wird.

Den ersten Fall dieser ODER-Verknüpfung nennt man, da man die Möglichkeit „sowohl Schwimmen als auch Surfen" ausschließt, auch ein ausschließendes oder exklusives ODER und verwendet XOR zur Unterscheidung dieser beiden OR-Varianten. Wird im Microsoft-BASIC ein OR verwendet, ist dies grundsätzlich das einschließende OR, also die Variante, die auch dann eine wahre Gesamtaussage liefert, wenn beide der geprüften Bedingungen wahr sind. Dies belegt auch die Wahrheitstafel.

Der dritte der logischen Operatoren NOT kehrt den Wahrheitsgehalt einer Aussage einfach um.

Wenn etwas NICHT wahr ist, ist es falsch. Daß hier auch der Rechner so vorgeht, können Sie leicht nachprüfen, indem Sie einfach

```
PRINT NOT -1
```

eingeben. Das Ergebnis ist, wie nicht anders erwartet, eine 0.

Logische Operationen werden beim Colour-Genie bitorientiert abgearbeitet. Damit unterscheidet sich das Microsoft-BASIC wesentlich von BASIC-Interpretern auf anderen Rechnern.

Dazu kurz zwei Beispiel (die Sie hoffentlich nicht zu sehr aus der Fassung bringen):

```
PRINT 2 OR 3
3
READY
>
```

und

```
PRINT 2 AND 3
2
READY
>
```

Diese recht seltsam erscheinenden Ergebnisse stiften eine Menge Verwirrung, wenn man sich nicht darüber klar ist, wie sie zustande kommen.

Dies ist jedoch sehr einfach:

2 OR 3

2 ist	00000010	in Dualschreibweise
Verknüpfung = OR		
3 ist	00000011	in Dualschreibweise
ergibt	00000011	= dezimal 3

Hier wurden Dualzahlen miteinander verknüpft, und zwar so, wie es die Wahrheitstafel vorschreibt. Analog gilt dies auch für die AND-Verknüpfung:

2 AND 3

2 ist	00000010	in Dualschreibweise
Verknüpfung = AND		
3 ist	00000011	in Dualschreibweise
ergibt	00000010	= dezimal 2

Das Verwirrende, nämlich, daß bei logischen Operationen, die eigentlich nur die Ergebnisse 0 oder −1 kennen, plötzlich Ergebnisse wie 2 oder 3 auftauchen, liegt also nur daran, daß das Ergebnis der logischen Verknüpfung zweier Zahlen eine Bitkette ist, die im Microsoft-BASIC als Dezimalzahl ausgegeben wird.

Natürlich ist es auch möglich, mehr als zwei Bedingungen in einer quasi „Mehrfachoperation“ zu prüfen:

```
PRINT 1 OR 7 OR 9 OR 11
```

Treten bei solchen Operationen die Operatoren AND, OR, NOT gemeinsam oder mehrfach auf, ist zu beachten, daß keinesfalls alle Operatoren die gleiche Abarbeitungspriorität besitzen. Wie bei arithmetischen Operationen die Klammerrechnung vor der Punktrechnung, die Punktrechnung vor der Strichrechnung steht, ist die Bindungsstärke der Operation NOT größer als die des AND, diese ist wiederum größer als die des OR.

Auf gut deutsch heißt das: Wenn Ihr Rechner die logische Operation

```
PRINT 0 OR 4 AND 0 OR 8
```

bestimmen soll, ist das das gleiche wie

```
PRINT 0 OR (4 AND 0) OR 8
```

Sollte dies nicht so ganz in Ihrem Sinne sein, müssen Sie durch das Setzen von Klammern Ihren (in dieser Beziehung ganz sturen) Partner Computer überzeugen.

Damit Ihnen hier keine Fehler unterlaufen, ist in Abb. 3.4 noch einmal eine vollständige Reihenfolgetabelle aufgeführt, aus der Sie die Bindestärke, d. h. die Abarbeitungspriorität der einzelnen Operatoren entnehmen können.

Wohl die geläufigste Variante logischer Operationen ist das Durchführen von Vergleichen. Hierfür stehen Ihnen die Zeichen < , > und = sowie die entsprechenden Kombinationen zur Verfügung. Wiederum ist das Ergebnis eines solchen Vergleichs ein logischer Wert, der entweder ein „Wahr" oder ein „Falsch" repräsentiert und zahlenmäßig eine −1 oder 0 liefert. Auch dieses ist beim Rechner leicht nachvollziehbar, indem Sie z. B.

```
PRINT 1=2
0
READY
>
```

eingeben. Hierzu sollten Sie Kapitel 13 Ihres Handbuchs aufschlagen, wo als Entscheidungsbefehl Vergleichsoperationen behandelt werden. Neu ist an dieser Stelle auch nur, daß hier logische Operationen nicht in der bekannten Form einer IF THEN-Anweisung auftauchen, sondern direkt in einer PRINT-Anweisung formuliert werden, was, wenn man gewohnt ist, das Gleichheitszeichen als Bestandteil einer Gleichung oder einer Wertzuweisung aufzufassen, einem irgendwie gegen den Strich geht.

Wahrscheinlich legt sich dieses Unwohlsein wieder, wenn Sie folgendes eingeben:

```
PRINT 1<2
-1
READY
>
```

Im ersten genauso wie im zweiten Beispiel führt der Rechner einen Vergleich der beiden Zahlen 1 und 2 durch. Nur die Vergleichsoperatoren sind andere: einmal =, einmal <.

Die Bedeutung des Gleichheitszeichens hängt also vom jeweiligen Zusammenhang ab. Während bei der Schreibweise B=2 der Variablen B der Wert 2 zugewiesen wird, soll bei der Schreibweise PRINT B=2 überprüft werden, ob die Variable B den Wert 2 tatsächlich hat.

Bindestärke der Operatoren	**Abarbeitungspiorität**
()	höchste
+ – NOT	!
^ [	!
* /	!
+ –	!
< <= > >= <> ><	!
AND	!
OR	niedrigste

Abb. 3.4: In welcher Reihenfolge Operationen durchgeführt werden, hängt von der Bindestärke der Operatoren ab.

3.3 VARIABLEN

Kapitel 4 Ihres Handbuchs behandelt die verschiedenen Variablentypen und deren Anwendung in BASIC. In diesem Kapitel neigt man als Anfänger dazu, zwar zu registrieren, daß es da eine Reihe verschiedener Varianten gibt, ohne sich jedoch darüber recht klar zu sein, wann und wo der Einsatz eines bestimmten Variablentyps von Vorteil sein kann oder wann er dringend erforderlich ist.

Besonders verwirrend für Ungeübte ist die Vielfalt der existierenden Notationen wie !, %, $, #, die verschiedenen Definitionsanweisungen wie DEFINT, DEFSNG, DEFDBL, DEFSTR und die Konvertierungsfunktionen CSNG, CDBL und CINT, durch die einmal definierte Variablentypen wieder in andere Typen umgewandelt werden können. Dazu ein Beispiel:

```
10 DEFSTR A
20 A="Hallo"
```

Wie Sie sehen, wird durch die DEFSTR-Anweisung dem Rechner mitgeteilt, daß alle folgenden Variablen, die mit dem Buchstaben A beginnen, Zeichenkettenvariablen sind. Die DEFSTR-Anweisung bietet also in erster Linie eine andere Schreibweise für Zeichenkettenvariablen, da kein $-Zeichen mehr an die Variable angehängt wird. In ähnlicher Form gilt dies auch für die DEFDBL-Anweisung. Lesen Sie hierzu bitte in Kapitel 24 auf Seite 72ff. Ihres Handbuchs nach.

Warum, meinen Sie, braucht man überhaupt so viele Variablentypen, wo doch zwei oder drei, vielleicht einer für numerische Variablen (A,AA,BC,C9) und einer für Zeichenkettenvariablen (A1$,B4$, etc.), vollständig ausreichen?

Die wichtigste Erklärung ist wohl die, daß Mikrocomputer wie das Colour-Genie vom verfügbaren Speicher her nicht so optimal bestückt sind, daß sie es sich leisten können, große Mengen an Speicherplatz einfach dadurch zu vergeuden, daß sie dem Rechner jeweils vier oder acht Bytes zur Speicherung eines Variablenwerts zubilligen, wo effektiv nur zwei Bytes benötigt werden.

Gerade bei längeren Programmen oder solchen, die mit Feldern arbeiten, kann eine solche Programmierung leicht Hunderte von Bytes beanspruchen, die anderswo dringend fehlen. Wie sich dies z. B. bei Feldern, die im folgenden Kapitel zur Sprache kommen, auswirkt, können Sie selbst nachvollziehen, indem Sie folgende beide Befehlsfolgen vergleichen:

```
NEW:PRINT MEM:DIMA(1000):PRINT MEM
```

einerseits und

```
NEW:PRINT MEM:DIMA%(1000):PRINT MEM
```

andererseits.

In beiden Fällen haben Sie den Rechner dazu veranlaßt, Speicherplatz für die Erfassung von 1000 Zahlenwerten zur Verfügung zu stellen. Wenn Sie wissen, daß in diesem Feld nur ganze Zahlen erfaßt werden, verschenken Sie, wenn Sie nicht durch das Prozentzeichen oder eine DEFINT-Anweisung die Variable als ganzzahlig bestimmen, glatte 2000 Bytes.

Das Prozentzeichen einer Variablen wird übrigens nicht mitgespeichert, sondern stellt nur eine zusätzliche Information für den Rechner dar, um welchen Variablentyp es sich handelt. Im Microsoft-BASIC ist es dem Benutzer freigestellt, eine derartige Typenkennung an Variablen anzuhängen, wenn diese im Programm verwendet werden, oder alternativ dazu am Anfang des Programms durch eine DEFINT, DEFSNG oder DEFSTR Anweisung global allen in einem Programm verwendeten Variablen, die mit bestimmten Anfangsbuchstaben beginnen, solche Typenmerkmale zuzuweisen.

Wollen Sie z. B. allen Variablen, die mit den Buchstaben A, B, C, D, L und M beginnen, das Attribut ganzzahlig zuweisen, so lautet die entsprechende Anweisung:

DEFINT A–D,L,M

Vermeiden Sie die Verwendung von ganzzahligen Variablen, wenn Sie Divisionen vornehmen!

```
10 DEFINT C
20 C=4/3
30 PRINT C
```

liefert das Ergebnis

```
1
READY
>
```

Doch nicht nur die Typenkennung einer Variablen ist von Bedeutung. Auch bei der Verwendung des Variablennamens sind bestimmte Vorschriften zu beachten. Die wichtigsten sind folgende:

1. Der Name einer Variablen muß immer mit einem Buchstaben beginnen.
2. Ein Variablenname kann beliebig lang sein, sofern in ihm keine BASIC-Schlüsselwörter enthalten sind.
3. Für das Microsoft-BASIC sind jedoch nur die ersten zwei Stellen eines Variablennamens signifikant. So sind z. B. für den Rechner die Variablen HAND, HASE und HA identisch.

Demzufolge sind z. B. die Variablen OTTO und KONTO unzulässig, da beide das BASIC-Schlüsselwort TO enthalten. In ähnlicher Weise gilt dies für Variablen wie NOTEN, FORMEL, RAND etc.

Bei der Verwendung von Variablen, die in ihrem Namen eine Aussage enthalten, ist also Vorsicht geboten!

Erwähnenswert im Zusammenhang mit Wertzuweisungen an eine Variable ist die Anweisung LET.

Will man die Wertzuweisung für eine Variable besonders betonen, kann man dies tun, indem man den Befehl LET vor die Variable setzt. Die Verwendung der LET Anweisung ist optional, d. h. es besteht kein Unterschied darin, ob Sie in einem Programm LET A=7 oder einfach A=7 schreiben. Bei Programmen allerdings, die hinterher compiliert werden sollen, d. h. in Maschinensprache übersetzt, spielt sowohl die LET-Anweisung als auch die Verwendung von ganzzahligen Variablen eine erhebliche Rolle. Ein compiliertes Programm, bei dem hauptsächlich ganzzahlige Variablen verwendet werden, vor die zusätzlich eine LET-Anweisung gesetzt wird, laufen in compilierter Fassung zwischen 40 und 60% schneller ab als solche Programme, in denen das nicht berücksichtigt wurde.

3.4 FELDER

Schon im vorherigen Abschnitt wurde kurz ein Feld definiert, um zu verdeutlichen, wie durch Verwendung von Variablen des richtigen Typs Speicherplatz gespart werden kann. Damit wurde, wenn auch nur als Mittel zum Zweck, diesem Kapitel vorgegriffen. Dort stand als Kernaussage

```
DIM A%(1000)
```

Die erste Auswirkung dieser Anweisung auf den Rechner, daß nämlich eine Menge Speicherplatz nach Betätigung der <RETURN>-Taste plötzlich nicht mehr zur Verfügung stand, haben Sie schon festgestellt. Dieser Platz wurde „reserviert“ für viele Variablen, die alle den Namen A% haben und zu ihrer Unterscheidung einen zusätzlichen sog. Index führen, der kennzeichnet, um welches A% es sich handelt. So können Sie z. B. nach Eingabe der o. a. DIM-Anweisung A%(501) mit dem Wert 100 belegen:

```
A%(501)=100: PRINT A%(501)
```

Alle übrigen A%-Variablen (mit dem Index 0 bis 500 und 502 bis 1000) werden durch diese Wertzuweisung nicht beeinflußt und behalten ihre bis dato zugewiesenen Werte.

Wozu benötigt man derartige Einteilungen in der Praxis? Die Fachliteratur sagt dazu: Immer dann, wenn ein Problem von einer oder mehreren Einflußgrößen abhängt und diese Größen in ganzen Zahlen, beginnend etwa mit 0 oder 1, ausgedrückt werden können.

Dazu ein Beispiel: Stellen Sie sich einen Wohnblock vor, bestehend aus mehreren gleichartig aufgebauten Hochhäusern. Über die dort wohnenden Familien wollen Sie Daten sammeln und später auswerten. Es interessiert Sie z. B. wie viele Personen jeweils in einem Haushalt leben.

Betrachten Sie zuerst eine Etage dieses Wohnblocks, z. B. die vierte. Sie wissen, daß dort, sagen wir, 8 Wohnungen existieren.

In Wohnung 1 wohnt eine sechsköpfige Familie, in Wohnung 2 ein verwitweter Rentner. Da 8 Wohnungen in Etage 4 zur Verfügung stehen, ist durch

```
DIM E4(8)
```

ein Feld hinreichend definiert.

E4(1)=6 steht für die sechsköpfige Familie
E4(2)=1 steht für den Rentner
...
E4(x)=? steht für die x-te Wohnung in dieser Etage.

Die Wohnungsnummer einer Etage ist also, um auf die o. a. Definition zurückzukommen, eine abhängige Einflußgröße, was die Anzahl der in einem bestimmten Haushalt lebenden Personen in der 4. Etage angeht.

Nun hat so ein Hochhaus in der Regel noch weitere Etagen. Gehen wir z. B. von 7 Etagen aus, benötigen wir zur Bestimmung eines ganz bestimmten Haushalts eine weitere Einflußgröße, die Etage, welche

Werte zwischen 1 und 7 annehmen kann. Ein Hochhaus HH% dieser Bauart ist durch

```
DIM HH%(7,8)
```

komplett in bestimmbare Wohnungseinheiten „zerlegbar“. Soll beispielsweise die Variable für die Etagennummer EN lauten und die für die Wohnungsnummer WN, dann kann die sechsköpfige Familie aus Etage 4 neu erfaßt werden durch

```
EN=4:WN=1:HH%(EN,WN)=6
```

Die Erweiterung eines solchen Feldes um weitere Einflußgrößen, wie mehrere Städe (ST) und mehrere Hochhäuser (HH), ist genausogut möglich. Sie könnten dann z. B. genau ermitteln, wie viele Personen in der Stadt 4, Hochhaus 3, Etage 4, Wohnungsnummer 8 leben. Ein solches 4-dimensionales Feld (vierdimensional deshalb, weil 4 Einflußgrößen eine bestimmte Datenmenge kennzeichnen) liegt jedoch weit außerhalb der Speicherkapazität eines Mikrocomputers, da sich die Anzahl der Variablen, für die Platz geschaffen werden muß, bei jeder neuen Einflußgröße um deren Maximalwert vervielfacht. Hinzu kommt, daß im Microsoft-BASIC des Colour-Genie ein Feldindex immer mit 0 beginnt, so daß ein Feld der Form

```
DIM A%(3)
```

Platz für 4 Feldelemente liefert, nämlich A%(0), A%(1), A%(2) und A%(3). Es ist in vielen Fällen jedoch verwirrend, das Feld mit dem Index 0 in Berechnungen und Erfassungen mit einfließen zu lassen.

Die Eintragung der einzelnen Elemente in ein Feld ist zwar recht mühsam, hat jedoch auch einige Vorteile. Sobald nämlich alle Angaben vollständig sind, kann man eine Vielzahl von Auswertungen durchführen, wie beispielsweise, wo Wohnungen leer stehen (ein 0-Personen-Haushalt), wo Großfamilien zu finden sind etc.

Genausogut können Sie ein weiteres Feld der Form

```
DIM HH$(7,8)
```

erstellen und dort Zeichenketten erfassen, wie

```
HH$(5,4)="Familie Schultz"
```

und so immer wissen, wer wo zu finden ist.

3.5 FUNKTIONEN

Zuerst eine Information vorweg, die für Colour-Genie Besitzer ohne Diskettenlaufwerk gilt.

Im Anhang finden Sie ein Maschinenprogramm, das auch unter Kassettenbetrieb das Arbeiten mit Funktionen erlaubt. Mit der DEF FN-Funktion, von der hier im folgenden die Rede ist, kann, wenn Sie nicht das im Anhang aufgeführte Treiberprogramm eingegeben und gestartet haben, sonst nur unter Disk-BASIC gearbeitet werden.

Dem Kapitel „Funktionen“ sollten Sie einen besonderen Stellenwert einräumen, da deren Anwendungsmöglichkeiten im Microsoft-BASIC vielseitig sind.

Eine Funktion stellt man sich immer sehr mathematikbezogen vor; schon allein Grund genug, einen großen Bogen darum zu machen. Schließlich quälen Lehrer ihre Schüler mit f(x)= 1/x und f(x)=sin(x) in der Regel so intensiv, daß Sie sicher keine große Lust verspüren, dies auf Ihr Hobby auszuweiten. Im Zusammenhang mit BASIC können Funktionen jedoch vollständig andere Bedeutungen erlangen. Dazu ein Beispiel:

Sie wollen ein Programm schreiben, das Vornamen, Namen und Telefonnummern über die Tastatur einliest und auf dem Bildschirm wie ein Telefonbuch ausgibt:

```
7 DEFFN S$(N1$,N2$,VW$,RN$)=N2$+", "+N1$
+"    "+VW$+"/"+RN$
10 INPUT"WIE LAUTET DER VORNAME :";V$
20 INPUT"WIE LAUTET DER NACHNAME :";N$
30 INPUT"WIE LAUTET DIE VORWAHL :";T1$
40 INPUT"WIE LAUTET DIE RUFNUMMER :";T2$

50 PRINT:PRINT PRINT FNS$(V$,N$,T1$,T2$)

READY
>
>RUN
WIE LAUTET DER VORNAME  ? HEINZ
WIE LAUTET DER NACHNAME :? BOLLE
WIE LAUTET DIE VORWAHL .? 0211
WIE LAUTET DIF RUFNUMMER  ? 1013


BOLLE. HEINZ      0211/1013
READY
>█
```

Abb. 3.5: Anwendungsbeispiel für Funktionen

Hier wurde eine Funktion in Zeile 7 definiert. Der Name dieser Funktion ist FN S$. Hinter dem Funktionsnamen stehen in Klammern, durch Kommata getrennt, die sogenannten abhängigen Variablen:

Die Variable für den Vornamen N1$
Die Variable für den Nachnamen N2$
Die Variable für die Vorwahl VW$
Die Variable für die Rufnummer RN$

Was bedeutet diese definierte Funktion für das Colour-Genie? Dem BASIC-Interpreter sagt diese Funktion folgendes:

Wenn ich irgendwann im Programm auf einen Funktionsaufruf der Form FN S$(...) treffe, so muß ich die dahinter in Klammern stehenden Variablenwerte ermitteln und diese so weiterverarbeiten, wie es mir der Definitionsteil der Funktion (DEFFNS$) vorschreibt.

Auf Zeile 7 bezogen bedeutet das, daß Nachname und Vorname vertauscht und durch ein Komma getrennt werden. Außerdem soll zwischen der Vorwahl und der Rufnummer ein / stehen.

Leicht nachvollziehbar? Wenn nein, geben Sie einfach einmal folgendes ein:

```
PRINT FNS$("Kalle","Anton","0999","44444")
```

und betätigen Sie die <RETURN>-Taste. Kennen Sie das Ergebnis schon? Es lautet

Anton, Kalle ... 0999/44444

Funktionen im Microsoft-BASIC treten also keinesfalls nur in der bekannten, mathematischen Form einer Funktionsgleichung auf. Vielmehr kann man sie überall dort einsetzen, wo Gleichungen und Zuweisungen übersichtlich gestaltet werden sollen, oder wenn in einem Programm eine bestimmte Gleichung häufig eingesetzt wird.

```
DEFN CUR=PEEK(&H4020)+256*PEEK(&H4021)-&H4400
```

ermittelt zum Beispiel jedesmal die gegenwärtige Position des Cursors auf dem Bildschirm, wenn Sie durch PRINT FNCUR eine entsprechende Berechnung wünschen. Aus diesem und dem vorherigen Beispiel sind einige der Vorteile von definierten Funktionen ableitbar:

* Bei mehrfacher Verwendung einer bestimmten Gleichung erweisen sie sich als platzsparend.
* Sie sind flexibel aufbaubar.
* Sie erlauben einen übersichtlichen Programmierstil.

* Man kann aus dem Namen leicht die Aufgabe, die die Funktion übernimmt, ableiten.

* Die im Definitionsbereich der Funktion verwendeten Variablen (wie z. B.: N1$, N2$, VW$, RN$ im Beispielprogramm in Zeile 7) üben keinen Einfluß auf gleichnamige Variablen im Programm aus. Wenn also irgendwo in Ihrem Programm die Variable A$ den Wert 12345 hat, ändert sich dieser Wert nicht. Er ist unabhängig davon, wie oft Sie Funktionen aufrufen, in deren Definitionsteil sich ein A$ als abhängige Variable befindet.

* Funktionen können verschachtelt sein, d. h. eine Funktion kann in ihrem Definitionsteil eine weitere aufrufen.

Ein Argument für die Anwendung von Funktionen ist es also, Formeln, die häufig in Programmen gebraucht werden, zusammenzufassen.

Besonders interessant sind solche Aufgaben, die sich nicht einfach durch Eingabe einer BASIC-Anweisung lösen lassen. Dazu gehört zum Beispiel die Ermittlung der PRINT @-Position, an der sich der Cursor gerade befindet. Die Funktion FNCUR, die Sie eben kennengelernt haben, leistet dies.

```
20 DEFFN TEST(X%)=(X%=100) ' ES SOLL UNT
ERSUCHT WERDEN, OB EINE ZAHL, Z B  Z%, D
EN WERT 100 HAT
30 INPUT"GEBEN SIE EINE ZAHL ZWISCHEN 99
 UND 101 EIN :";Z%
35 '


40 IF FNTEST(Z%)=0 THEN PRINT"WERT <>100
" ELSE PRINT"WERT=100"
READY
>
>
>RUN
GEBEN SIE EINE ZAHL ZWISCHEN 99 UND 101
EIN :? 100
WERT=100
READY
>█
```

Abb. 3.6: Mit Hilfe einer Funktion wird eine Bedingung geprüft

Einmal definiert, arbeitet FNCUR wie alle anderen, „offiziellen" Funktionen im Microsoft-BASIC. Es reizt Sie nun sicherlich, weitere Funktionen dieser Art kennenzulernen, um die Leistungsfähigkeit Ihres Interpreters zu erweitern. Bevor wir allerdings detailliert auf dieses Thema eingehen können, möchten wir Ihnen an einem Beispiel eine weitere Einsatzform von Funktionen vorstellen (siehe Abb. 3.6).

Geben Sie dieses Programm ein, und starten Sie es. Geben Sie abwechselnd für Z% Werte gleich oder ungleich 100 ein. Warum, meinen Sie, läuft dieses Programm?

Lassen Sie uns Schritt für Schritt die einzelnen Operationen des Rechners zurückverfolgen. Das kann man am besten in der Kommandoebene:

```
NEW <RETURN>
```

Erster Schritt:

```
X=(100=100):PRINTX <Return>
-1
READY
>
```

Zweiter Schritt:

```
IF (100=100)=-1 THEN PRINT"NATUERLICH IST 100 = 100" <Re-
turn>
NATUERLICH IST 100 = 100
READY
>
```

Vergleichen Sie beide Schritte. Was dem Menschen natürlich erscheint, ist auch für den Rechner eine Selbstverständlichkeit. Der Unterschied ist nur, daß beide sich unterschiedlich ausdrücken. Wenn für das Colour-Genie bei einem Vergleich sich etwas als „wahr" erweist, „merkt" es sich eine −1. Stellt es bei einem Vergleich ein „Falsch" fest, „merkt" es sich eine 0. (Vergleichen Sie dazu auch das Kapitel „Operationen".)

Beim ersten wie beim zweiten Schritt wurde, obwohl es durch die Art der Darstellung für Sie fremdartig aussehen mag, durch (100=100) der Rechner veranlaßt, einen Vergleich durchzuführen. Das Ergebnis dieses Vergleichs war, daß es sich um eine „logisch (durchaus) wahre" Aussage handelte.

Der Unterschied zwischen den beiden Schritten besteht also einzig und allein darin, daß in Schritt 1 das Vergleichsergebnis −1 einer Variablen, nämlich X zugewiesen wird, während der Rechner in Schritt 2 anhand des Ergebnisses sich „entscheidet", einen Text auszugeben.

Ist Ihnen das Prinzip klar geworden? Trifft der Rechner auf ein

```
X= (100=100) oder X=(100>100) oder allgemein X= (A>B)
```

so faßt er dies als eine Aufforderung auf, den Ausdruck in Klammern auf seine Wahheit hin zu prüfen. Eine solche „Aufforderung" kann auch Bestandteil einer definierten Funktion sein. Dies können Sie leicht auf die o. a. Beispiele anwenden, indem Sie Zeile 20 unseres Beispielprogramms wie folgt abändern:

```
20 DEFFN TEST (X%)=(X%>=100)
```

und den ausgegebenen Text in Zeile 40 entsprechend der geprüften Bedingung anpassen:

```
40 IF FNTEST (Z%)=0 THEN PRINT"ZAHL<100" ELSE PRINT"ZAHL
>= 100"
```

Wenn Sie sicher sind, daß Sie diese Grundlagen verstanden haben, können wir uns dem nächsten Kapitel zuwenden, wo Sie Anwendungsbeispiele für eigene Funktionen finden und sehen, wie Sie diese aufbauen und wo Sie sie einsetzen können.

3.6 EIGENE FUNKTIONEN

Dieses Kapitel ist etwas schwieriger als die vorherigen, zumal weder im Colour-BASIC-Handbuch noch in der Disk-BASIC-Anleitung besonders viel darüber aufgeführt ist.

Daher vorweg eine kurze Beschreibung, wie Sie Funktionen im Microsoft-BASIC definieren und diese aufrufen:

Das Schlüsselwort, durch das der Rechner erkennt: Hier wird eine Funktion definiert, lautet DEFFN. Jede Funktion hat einen Namen, der mindestens zwei Zeichen lang sein muß, wie z. B.: AA$,AAA$ (wenn es sich um eine Zeichenkettenfunktion handelt) oder A1,AA (wenn es sich um eine numerische Funktion handelt). Wenn der Funktion bei ihrem Aufruf keine variablen Werte übergeben werden sollen (s. a. Kapitel „Funktionen"), folgt das Gleichheitszeichen und die Funktion, wie z. B.:

```
10 DEFFN BS$=CHR$(PEEK(&H4400))
RUN
```

die immer, wenn Sie

```
PRINT FNBS$
```

eingeben, das Zeichen ausgibt, das sich in der linken oberen Ecke auf dem Bildschirm befindet.

Genie-BASIC solche Zahlenkonvertierungen, wie z. B. die Umwandlung einer Hexadezimalzahl in eine Dezimalzahl:

```
PRINT &H4400 <RETURN>
17408
READY
>
```

Den umgekehrten Weg, dezimal in hex, unterstützt das Microsoft-BASIC des Colour-Genie allerdings nicht. Grund genug, diesen Mangel durch ein kurzes Programm zu beheben:

```
10    ' DAS FOLGENDE PROGRAMM DEFINIERT DIEFUNKTIO
      NEN FN HX$(X%) UND FN HEX$(X%)   ALS KONVERT
      IERUNGSFUNKTIONEN EINER DEZI-MALZAHL IN EINE
       HEXADEZIMALZAHL
20    CLEAR 1000
30    '*** ANFANG DES DEFINITIONSTEILS ****
40    H0$="0123456789ABCDEF"
50    DEFFN HX$(X%)=MID$(H0$,INT(X%/16)+1,1) + MID
      $(H0$,X%-INT(X%/16)*16+1,1):'FUER ZAHLEN VON
       0 BIS 255
60    DEFFN HEX$(X%)=FN HX$(INT(X%/256)) +FNHX$(X%
       AND 255) :' FUER ZAHLEN VON 0  BIS 32767
70    '*** ENDE DES DEFINITIONSTEILS *****
80    INPUT"WIE LAUTET DIE DEZIMALZAHL :";X%
90    PRINT"DIE DEZIMALZAHL ";X%;" ENTSPRICHT HEXA
      DEZIMAL ";
100   IF X%>255 THEN PRINTFN HEX$(X%):GOTO80
110   PRINTFN HX$(X%):GOTO80
```

Abb. 3.7: Konvertierungsfunktionen (dezimal in hex)

Es gibt viele Aufgaben, die sich effizienter lösen lassen, wenn man von vornherein mit Hexadezimalzahlen rechnet. Außerdem ist die FNHEX$(x)-Funktion recht nützlich, wenn Sei die vielen in Ihrem BASIC-Handbuch aufgeführten Dezimaladressen im Speicherbelegungsplan auf Seite 124 einordnen wollen. Wenn Sie dieses Programm starten, bekommen Sie ungefähr folgende Bildschirmausgabe:

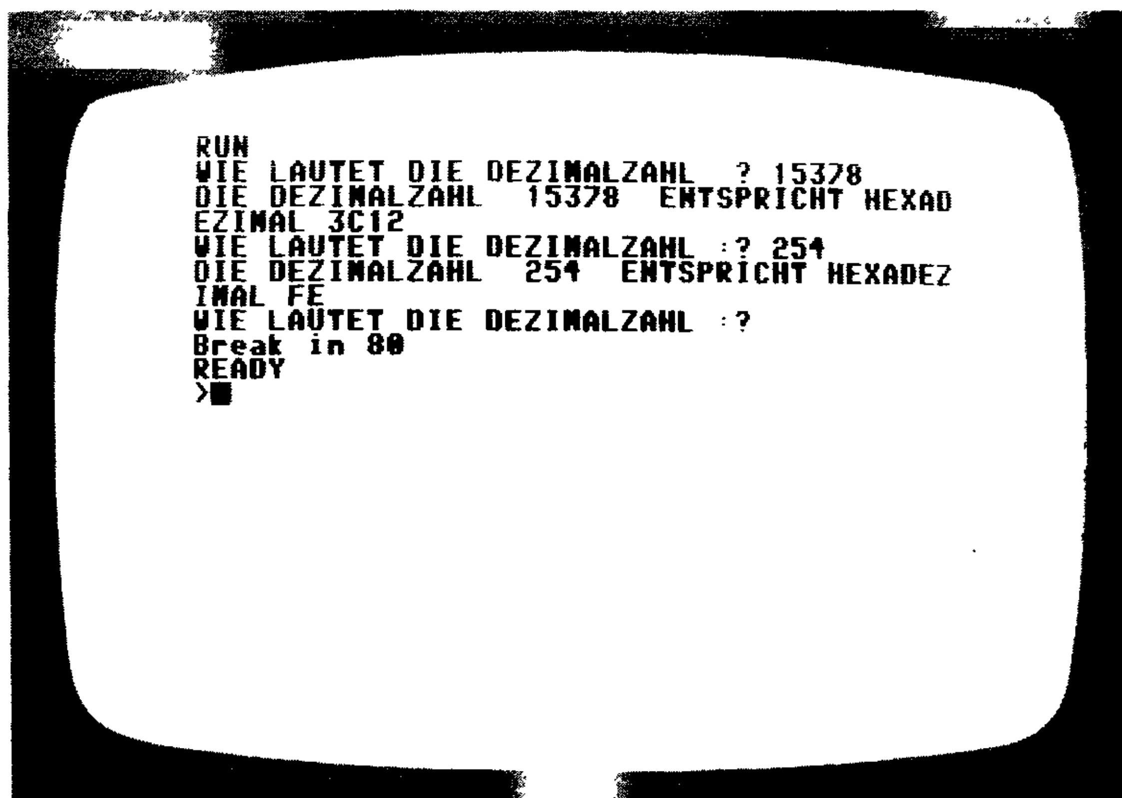


Abb. 3.8: Bildschirmausgabe zum Programm in Abb. 3.7

Schauen Sie sich das Programm genau an, und Sie werden feststellen, daß dort zwei Funktionen definiert wurden. Die eine, FN HX$(X%), wird in Zeile 110 immer dann aufgerufen, wenn die eingegebene Dezimalzahl kleiner als 256 ist. Dies ist keineswegs unbedingt notwendig, denn auch bei Zahlenwerten, die kleiner sind als 256, besitzt die zweite definierte Funktion, FN HEX$(X%), ihre Gültigkeit. Vielmehr sehen Sie hier das erstemal ineinander verschachtelte Funktionen, denn im Definitionsteil von HEX$(...) wird FN HX$(...) aufgerufen. Um sich die prizipielle Arbeitsweise dieser Funktion zu verdeutlichen, schlagen Sie am besten Kapitel 31 auf Seite 104 Ihres Handbuchs auf. Dort finden Sie ein Programm, das 17 Zeilen benötigt, um weitaus unkomfortabler die Dezimal-Hex-Konvertierung vorzunehmen. Hier sieht man sehr schön, wie effizient, kurz und elegant sich Funktionen einsetzen lassen.

3.6.1 Auslesen eines frei definierten Zeichens

Eine weitere Form der Konvertierung, die beim Umgang mit Rechnern früher oder später wünschenswert ist, ist die Konvertierung von Dezimal in Dual. Ein Beispiel dafür sind die in Ihrem Rechner frei programmierbaren 128 Zeichen. Was nützt es Ihnen, wenn Sie zum Beispiel beim Aus-

lesen des dafür zuständigen Speicherbereichs folgende Werte geliefert bekommen:

```
100 REM AUSLESEN DES SPEICHERS FÜR DAS ERSTE FREI
        PROGRAMMIERBARE ZEICHEN AB SPEICHERADRESSE
        HF400
110 FOR X=&HF400 TO &HF407
160 PRINT PEEK(X)
170 NEXT X
```

Wenn Sie das Beispiel im Colour-Genie-Handbuch auf Seite 110 eingegeben haben, bekommen Sie folgende Bildschirmausgabe:

```
146
40
68
68
124
68
0
READY
>
```

Dadurch wissen Sie allerdings noch lange nicht, wie das definierte Zeichen aussieht. Wir definieren also eine entsprechende Funktion, die wir in das oben aufgeführte Programm einbauen:

```
10      'FUNKTION ZUM FESTSTELLEN GESETZTER
            BZW. NICHT GESETZTER BITS EINER ZAHL
20      '*** ANFANG DES DEFINITIONSTEILS****
30      DEFFN BIT(Z%,X%)=SGN(ABS(PEEK(Z%) AND (2[X%)
        ))
40      '*** ENDE DES DEFINITIONSTEILS ******
50      GOTO 100 :'UEBERSPRINGEN DES UNTER-
                    PROGRAMMS
60      FOR T%=7 TO 0 STEP -1
70      PRINT FNBIT(X,T%);" ";
80      NEXT T%
90      RETURN
100     X=&HF400:GOSUB 60 :'BESTIMMEN UND
                              AUSGEBEN DER
                              GESETZTEN BITS
```

Abb. 3.9: Welche Bits sind gesetzt?

Wird die Funktion in Zeile 30 (DEFFN BIT(Z%,X%)) aufgerufen, so müssen für Z% der zu untersuchende Zahlenwert und für X% das zu testende Bit übergeben werden. Das Unterprogramm ab Zeile 60 testet über eine Schleife alle 8 Bits der untersuchten Zahl. Ist das Bit gesetzt, wird eine 1 ausgegeben, bei zurückgesetztem Bit eine 0. Alle Einsen entsprechen dunklen Bildpunkten, alle Nullen hellen.

Nun ist es leicht, sich alle definierten oder z. B. das 80. Zeichen auf dem Bildschirm im Großformat ausgeben zu lassen. Schauen Sie zuvor noch einmal in Ihr BASIC-Handbuch auf Seite 110.

Dort können Sie feststellen, daß jedes frei definierbare Zeichen ein festes Raster, eine sog. Matrix besitzt, durch die es in Länge und Breite begrenzt wird. Das erste Zeichen ist ab Adresse &HF400, das zweite ab &HF408 etc. abgelegt. Wenn Sie also das n-te Zeichen ausgegeben haben wollen, läßt sich die dazugehörige Adresse leicht berechnen:

Adresse = (n−1)*8 + &HF400

Eine entsprechende Funktion lautet also

```
35 DEFFN ADR(N%)=(N%−1) * 8 +&HF400
```

und eine dazugehörende INPUT-Anweisung:

```
10    'FUNKTION ZUM FESTSTELLEN GESETZTER
          BZW. NICHT GESETZTER BITS EINER ZAHL
20    '*** ANFANG DES DEFINITIONSTEILS****
30    DEFFN BIT(Z%,X%)=SGN(ABS(PEEK(Z%) AND (2[X%)
      ))
35    DEFFN ADR(N%)=(N%-1)*8+&HF400
40    '*** ENDE DES DEFINITIONSTEILS ******
50    GOTO 100 :'UEBERSPRINGEN DES UNTER-
                     PROGRAMMS
60    FOR T%=7 TO 0 STEP -1
70    PRINT FNBIT(X,T%);" ";
80    NEXT T%
90    RETURN
100   '
110   INPUT"DAS WIEVIELTE FREI DEFINIERBARE ZEICHE
      N ";N%
111   IF N%>128 THEN PRINT"ES GIBT NUR 128 SOLCHER
       ZEICHEN":GOTO 110
112   FOR X=FN ADR(N%) TO FN ADR(N%)+7
113   GOSUB 60:PRINT:PRINT
114   NEXTX
```

Abb. 3.10: Auslesen eines Zeichens

Schon können Sie sich alle definierten Zeichen auf dem Bildschirm ausgeben lassen. In Abb. 3.11 finden Sie eine entsprechende Bildschirmausgabe.

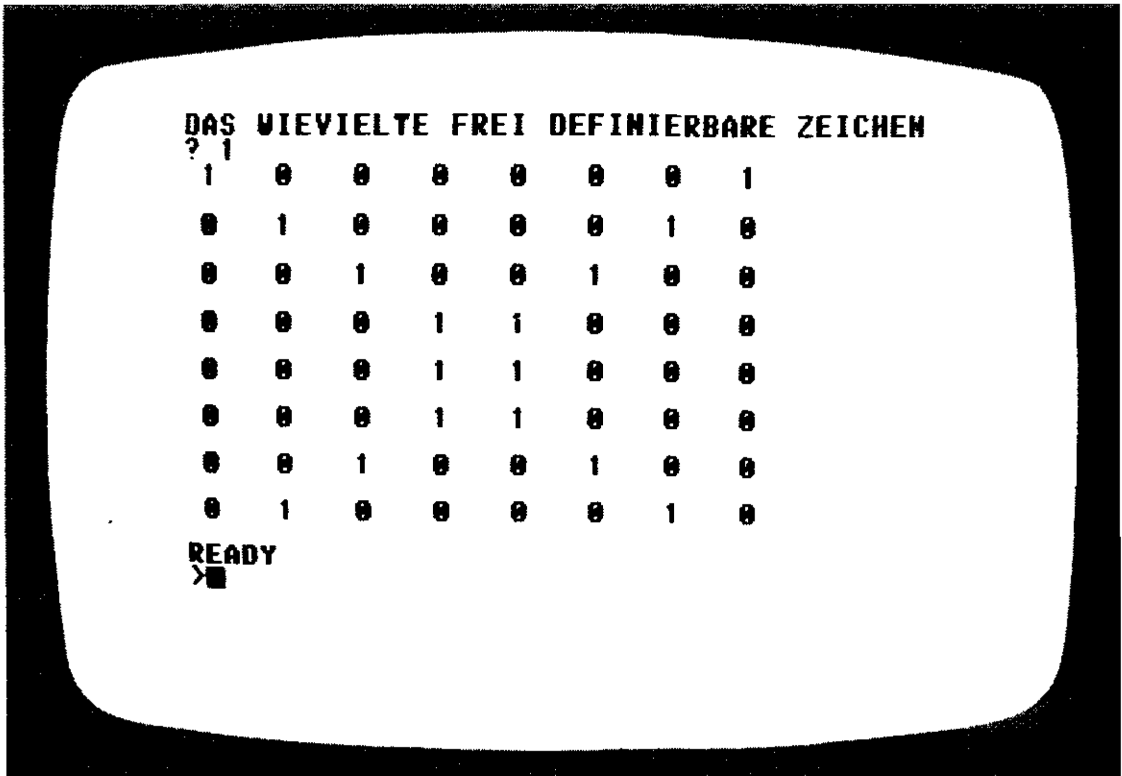


Abb. 3.11: Bildschirmausgabe zum Programm in Abb. 3.10

Von Konvertierungsroutinen kommen wir nun zu einer etwas interessanteren Form von definierten Funktionen.

Sicher kennen Sie das Problem, daß Sie mit einem Bekannten am Telefon einen Termin ausmachen wollen, der irgendwann auf die nächsten zwei oder drei Wochen datiert ist.

Die obligatorische Frage, die dann meist am einen oder anderen Ende der Leitung gestellt wird, lautet: „Was für ein Wochentag ist das?“ oder „Wie viele Tage sind es noch bis dahin?“

In der Regel beginnt dann die erfolglose Suche nach einem Kalender oder kompliziertes Kopfrechnen, so daß Sie die Möglichkeit, ein solches Problem vom Rechner lösen zu lassen, sicher mit erheblichem Arbeitsaufwand und einem langen Programm in Verbindung bringen. Dies ist kei-

```
50    DEF FN DA(TT, MM, JJ)=NOT((TT<1) OR (TT>31)
      OR (MM<1) OR (MM>12)  OR ((TT>30) AND ((MM=2
      ) OR (MM=4) OR (MM=6) OR (MM=9) OR (MM=11)))
       OR ((TT>29) AND (MM=2)) OR ((TT=29) AND ((M
      M=2) AND ((JJ/4)<> INT(JJ/4)))))


80    DEF FN WT(TT, MM, JJ)=JJ*365+INT((JJ-1)/4)+(
      MM-1)*28+VAL(MID$("000303060811131619212426"
      ,(MM-1)*2+1, 2))-((MM>2)AND((JJ AND NOT -4)=
      0))+TT
```

Abb. 3.12: Funktion zur Ermittlung eines Zeitraums

nesfalls so. Es gibt eine Funktion, die die ganze Problematik von Schaltjahren, Monaten mit 31, 30 und 28 Tagen von 1901 bis 2099 erfaßt hat. In ähnlicher Form findet diese Funktion in Digitaluhren Anwendung. Wenn Sie das Programm mit dieser Funktion starten und anschließend

```
PRINT FN WT(17,12,1983)
```

eingeben, ermittelt diese Funktion die Anzahl der Tage seit dem 1.1.0000:

```
724641
READY
>
```

Daraus läßt sich eine Menge ableiten: die noch verbleibenden Tage bis Weihnachten (das ist die Differenz zwischen zwei Terminen, nämlich dem 24. 12. und einem weiteren Datum) oder, mit Hilfe einer weiteren Funktion, der Wochentag eines beliebigen Datums:

```
20 DEFFN T$(Z!)=MID$("FrSaSoMoDiMiDo",
             (Z!-INT(Z!/7)*7)*2+1,2)
```

Nun wird der Rechner, wenn Sie wissen wollen, welcher Wochentag der 31. 10. 1983 war und

```
PRINT FN T$(FN WT(31,10,1983))
```

eingeben

```
Mo
READY
>
```

ausgeben. Wie Sie diese Funktion in eigenen Programmen anwenden können, sollen die Beispiele in Abb. 3.13 bis Abb. 3.16 verdeutlichen.

```
10    CLEAR 2000
20    'Anwendungsbeispiel:
       Arbeiten mit Funktionen
30    'DAS LEBENSALTER IN TAGEN
40    '
50    DEF FN DA(TT, MM, JJ)=NOT((TT<1) OR (TT>31)
      OR (MM<1) OR (MM>12)  OR ((TT>30) AND ((MM=2
      ) OR (MM=4) OR (MM=6) OR (MM=9) OR (MM=11)))
       OR ((TT>29) AND (MM=2)) OR ((TT=29) AND ((M
      M=2) AND ((JJ/4)<> INT(JJ/4)))))
60    '-1 bedeutet: Es gibt ein solches Datum
70    ' 0 bedeutet: Das Datum gibt es nicht
80    DEF FN WT(TT, MM, JJ)=JJ*365+INT((JJ-1)/4)+(
      MM-1)*28+VAL(MID$("000303060811131619212426"
      ,(MM-1)*2+1, 2))-((MM>2)AND((JJ AND NOT -4)=
      0))+TT
90    'Anzahl Tage seit 1.1.0000
91    DEF FNT$(Z!)=MID$("FrSaSoMoDiMiDo",(Z'-INT(Z
      !/7)*7)*2+1,2)
92    'Wochentag
110   INPUT"Bitte Eingabe
      des Tagesdatums(TTMMJJJJ) :";T$
120   IF LEN(T$)<>8 THEN 110
130   GOSUB 150
140   GOTO 210
150   'Ermittlung von Tag,Monat,Jahr
160   TT=VAL(LEFT$(T$, 2))
170   MM=VAL(MID$(T$, 3, 2))
180   JJ=VAL(RIGHT$(T$, 4))
200   RETURN
210   IF FN DA(TT, MM, JJ)=0 THEN PRINT"Ein solche
      s Datum gibt es nicht.":FORT=1TO1000:NEXTT:R
      UN
220   HEUTE=FN WT(TT, MM, JJ)
230   INPUT"Bitte das Geburts-
      datum(TTMMJJ) :";T$
240   GOSUB 150
245   IF FN DA(TT, MM, JJ)=0 THEN PRINT"Ein solche
      s Datum gibt es nicht.":FORT=1TO1000:NEXTT:R
      UN
250   GEBURT=FNWT(TT,MM,JJ)
260   ALTER=HEUTE-GEBURT
270   IF (ALTER/365)>90 THEN PRINT"Nun kommen Sie!
       Das nimmt Ihnen keiner ab."
280   PRINT"Sie sind ";ALTER;" Tage alt."
```

Abb. 3.13: Programm zur Berechnung des Lebensalters

```
Bitte Eingabe
des Tagesdatums(TTMMJJJJ) :? 30021984
Ein solches Datum gibt es nicht.
Bitte Eingabe
des Tagesdatums(TTMMJJJJ) :? 30041984
Bitte das Geburts-
datum(TTMMJJ) :? 15011958
Sie sind  9602  Tage alt.
READY
>█
```

Abb. 3.14: Bildschirmausgabe zum Programm in Abb. 3.13

```
10    CLEAR 2000
20    'Anwendungsbeispiel: Arbeiten mit Funktionen

30    '                      Ermittlung des Tagesdat
      ums
40    '
50    DEF FN DA(TT, MM, JJ)=NOT((TT<1) OR (TT>31)
      OR (MM<1) OR (MM>12)  OR ((TT>30) AND ((MM=2
      ) OR (MM=4) OR (MM=6) OR (MM=9) OR (MM=11)))
       OR ((TT>29) AND (MM=2)) OR ((TT=29) AND ((M
      M=2) AND ((JJ/4)<> INT(JJ/4)))))
60    '-1 bedeutet: Es gibt ein solches Datum
70    ' 0 bedeutet: Das Datum gibt es nicht
80    DEF FN WT(TT, MM, JJ)=JJ*365+INT((JJ-1)/4)+(
      MM-1)*28+VAL(MID$("000303060811131619212426"
      ,(MM-1)*2+1, 2))-((MM>2)AND((JJ AND NOT -4)=
      0))+TT
90    'Anzahl Tage seit 1.1.0000
91    DEF FNT$(Z!)=MID$("FrSaSoMoDiMiDo",(Z!-INT(Z
      !/7)*7)*2+1,2)
92    'Wochentag
100   CLS
110   INPUT"Bitte Eingabe des Tagesdatums(TTMMJJJJ
      ) :";T$
120   IF LEN(T$)<>8 THEN 110
```

Abb. 3.15: Programm zur Ermittlung des Wochentags

```
130   GOSUB 150
140   GOTO 210
150   'Ermittlung von Tag,Monat,Jahr
160   TT=VAL(LEFT$(T$, 2))
170   MM=VAL(MID$(T$, 3, 2))
180   JJ=VAL(RIGHT$(T$, 4))
200   RETURN
210   IF FN DA(TT, MM, JJ)=0 THEN PRINT"Ein solche
      s Datum gibt es nicht.": FORT=1TO1000:NEXTT:
      RUN
220   HEUTE=FN WT(TT, MM, JJ)
230   T$=FNT$(HEUTE)
240   PRINT"Heute haben wir ";T$;" ,den";TT;".";MM
      ;".";JJ
```

Abb. 3.15: Programm zur Ermittlung des Wochentags (Fortsetzung)

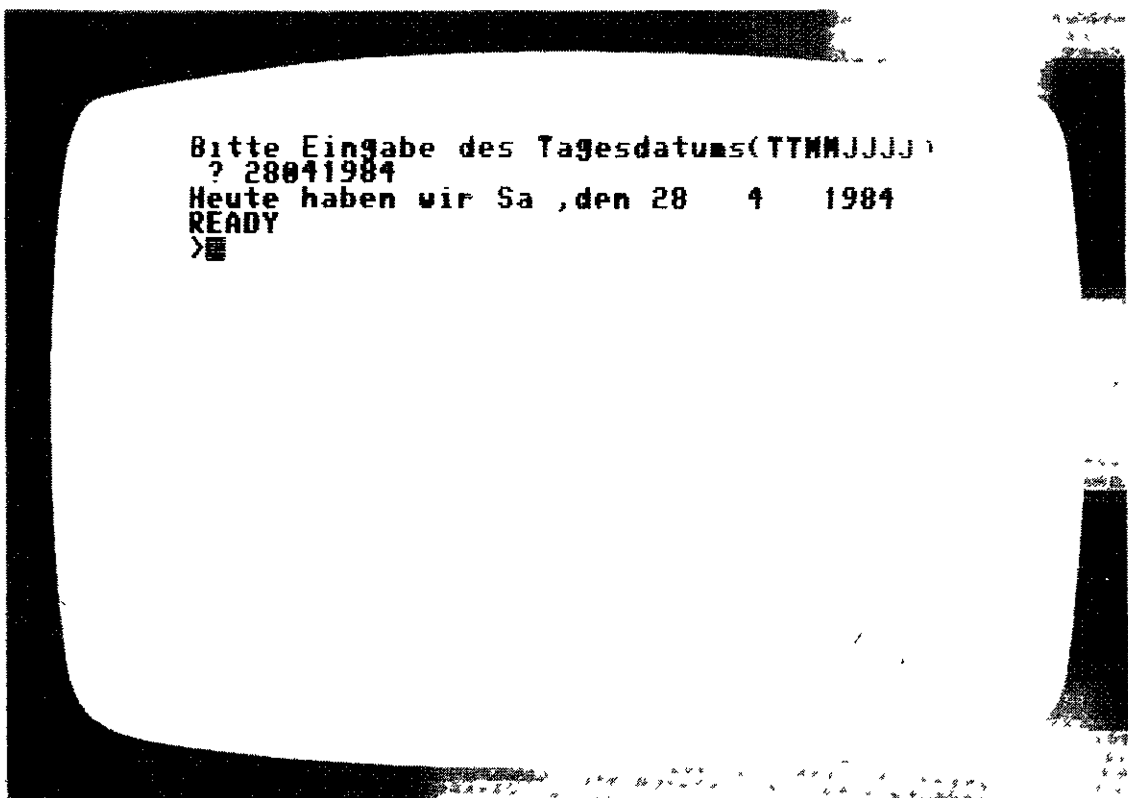


Abb. 3.16: Bildschirmausgabe zum Programm in Abb. 3.15

Zu diesem Zeitpunkt ist allerdings noch nicht gewährleistet, daß, wenn Sie das Programm in Abb. 3.11 starten, auch ein Datum eingegeben wird, das existiert. Ihr Colour-Genie berechnet den 31.2.1811 genauso wie den 66.1.1999 – nur falsch.

Bevor Sie diese Problematik in Form einer Funktion erfassen und so derart unsinnige Eingaben verhindern können, müssen Sie sich erst einmal Gedanken darüber machen, wann ein Datum zulässig ist und wann nicht.

Zuerst einmal ist ein Datum unsinnig, wenn der Wert TT für den Tag kleiner als 1 oder größer als 31 ist:

(TT < 1) OR (TT > 31)

Das gleiche gilt für den Monat MM, wenn hier Werte kleiner als 1 oder größer als 12 auftauchen:

(MM < 1) OR (MM > 12)

Weiterhin haben der April, Juni, September etc. nur jeweils 30 Tage. Dies kann folgendermaßen erfaßt werden. Das Datum gibt es nicht, wenn der Tag TT größer ist als 30 und gleichzeitig der Monat MM = 2 oder 4 oder 6 oder 9 oder 11, also

((TT> 30) AND ((MM=2) OR (MM=4) OR (MM=6) OR (MM=9) OR (MM=11))

Nun folgt noch der verflixte Februar, der Monat MM=2. Hier liegt immer dann ein Schaltjahr mit 29 Tagen vor, wenn die Jahreszahl JJ ohne Rest durch 4 teilbar ist. Ist dies nicht der Fall, dann ist

(JJ/4) ungleich INT(JJ/4)

das Datum also als unzulässig anzusehen, wenn

a) (TT>29) AND (MM=2)
b) ((TT=29) AND ((MM=2) AND ((JJ/4) <> INT(JJ/4))))

All diese Bedingungen, miteinander verknüpft, ergeben die gesuchte Funktion

```
50 DEF FN DA(TT,MM,JJ) = NOT((TT<1) OR (TT>31) OR (MM<1) OR
(MM>12) OR ((TT>30) AND ((MM=2) OR (MM=4) OR (MM=6) OR
(MM=9) OR (MM=11))) OR ((TT>29) AND (MM=2)) OR ((TT=29)
AND ((MM=2) AND ((JJ/4)<>INT(JJ/4)))))
```

Diese Funktion finden Sie in den Beispielen Abb. 3.11 bis 3.14. Sie ist eine komplizierte Variation dessen, was Ihnen unter Kapitel „Eigene Funktionen" als Funktion zur Prüfung logischer Bedingungen vorgestellt wurde. Aus dem Ergebnis, 0 oder ungleich 0, kann hier direkt abgelesen werden, ob ein untersuchtes Datum existiert oder nicht:

```
IF (FN DA(TT,MM,JJ)) THEN GOTO ....: 'EINGABEFEHLER
```

Lassen Sie sich durch die kompliziert erscheinenden Definitionsteile der Funktionen nicht irritieren. Definieren Sie sie irgendwo am Anfang Ihres Programms, und verwenden Sie diese Funktionen genauso wie andere,

die Ihnen im Microsoft-BASIC zur Verfügung stehen, sei es MID, ATN oder SQR, über deren komplizierte Abarbeitungsphasen im Interpreter Sie sich bestimmt auch nicht sonderlich viele Gedanken machen. Eine definierte Funktion solcher Komplexität wie FN WT oder FN DA soll als Hilfsmittel, als Mittel zum Zweck auf dem Weg des Programmierens dienen. Es reicht, wenn Sie diesem Kapitel die Anregung entnehmen, daß es da im BASIC etwas gibt, was DEF FN heißt, womit sie experimentieren und vielleicht interessante Effekte in eigenen Programmen erzielen können. In diesem Sinne sollen auch die in diesem Kapitel vorgestellten Funktionen und Demoprogramme zu sehen sein: Als Ergebnisse des Experimentierens.

Kapitel 4

Eine generelle Eingaberoutine zur Textverarbeitung

Die Fähigkeit zur Verarbeitung von Zeichenketten ist eine wesentliche Ursache für die Leistungsfähigkeit von Mikrocomputern. Rechner der früheren Generation, die mit Programmiersprachen wie ALGOL arbeiteten, verfügten nur eingeschränkt über entsprechende Algorithmen und engten daher ihre Einsatzmöglickeit selbst erheblich ein.

Obwohl mit heutigen Rechnern wie dem Colour-Genie derartige Probleme der Vergangenheit angehören, heißt dies noch lange nicht, daß das „allwissende Universalgenie" Rechner all Ihre Eingaben registrieren oder auf Fragen Patentlösungen anbieten kann.

Versuchen Sie doch einfach einmal, Ihrem Colour-Genie den Spielstand des Fußballspiels „Arminia Bielefeld gegen Bayern München" zu vermitteln:

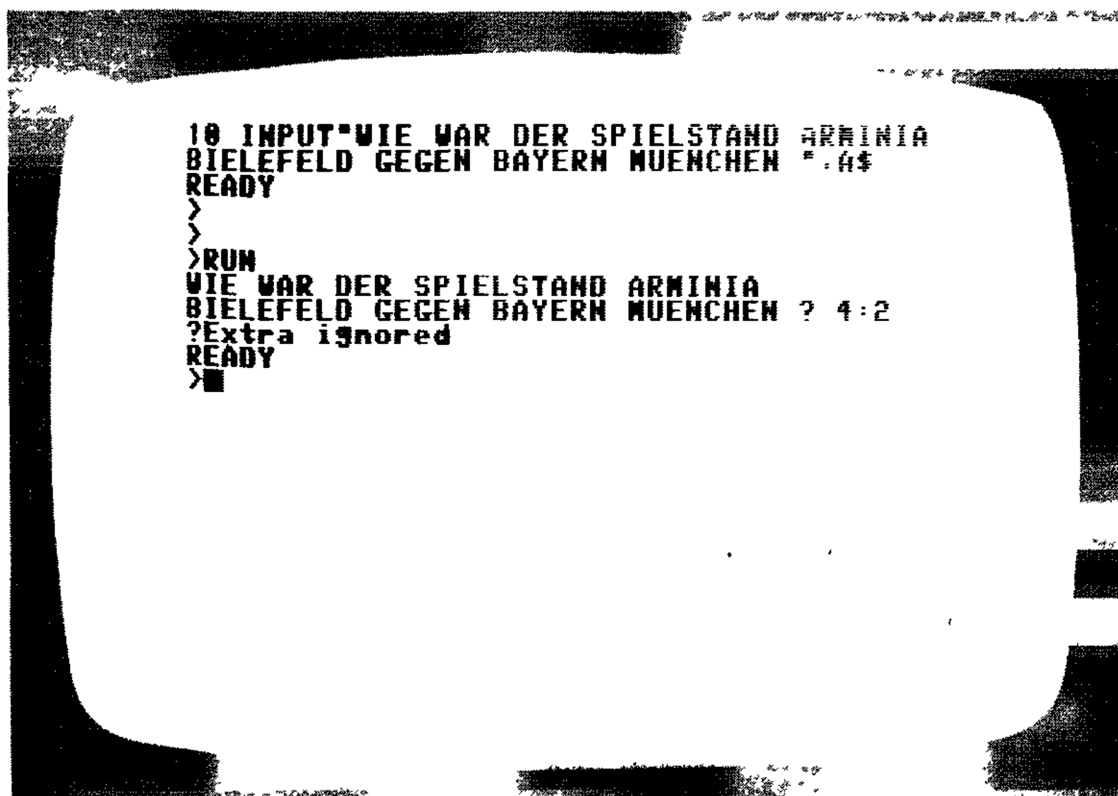


Abb. 4.1: Eingabeprobleme bei der INPUT-Anweisung

Beantworten Sie diese Frage mit einer beliebigen Eingabe wie 4:2 oder 1:3, und beachten Sie das Ergebnis. Eine entsprechende Bildschirmausgabe finden Sie in Abbildung 4.1.

Da kaum zu erwarten ist, daß Ihr Colour-Genie gegen den einen oder anderen Verein eine spezielle Antipathie hegt und aus diesem Grunde durch ein „?Extra ignored“ lauthals seinen Protest gegen die Eingabe kundtut, scheint es Zeichen zu geben, die auch beim Einlesen einer Zeichenkette nicht vorbehaltlos übernommen werden. Dies bestätigt auch das Colour-Genie-Handbuch auf Seite 29:

„Ein nach einer INPUT-Aufforderung eingegebener Doppelpunkt bewirkt in jedem Fall, daß alles, was danach kommt, nicht übernommen wird.“

Das Problem, einer Zeichenkettenvariablen auch solche Zeichen wie , ; : in einer Eingabe zuzuweisen, läßt sich nicht allein durch die INPUT-Anweisung lösen. (Es gibt zwar eine LINE INPUT-Anweisung, bei der dieses Problem nicht auftritt. Sie ist jedoch nur unter Colour-Disk-BASIC verfügbar.)

Eine Alternative findet sich in der INKEY$-Funktion, die auf Seite 96 Ihres Handbuchs beschrieben ist. Immer, wenn das Colour-Genie auf ein INKEY$ trifft, fragt es für Sekundenbruchteile die Tastatur ab, ob dort eine Taste betätigt wurde:

```
10 Y$=INKEY$
```

Ist dies der Fall, wird der Variablen Y$ eine Zeichenkette mit der Länge 1 zugewiesen, die dem ASCII-Zeichen der betätigten Taste entspricht. Wurde zum Zeitpunkt der Abfrage keine Taste betätigt, erhält die Variable Y$ keine Information. Man sagt, Y$ enthält eine leere Zeichenkette, was nichts anderes bedeutet als

```
Y$ = ""
```

Um nun sicherzugehen, daß ein Zeichen von der Tastatur eingelesen wird, muß so lange eine Schleife durchlaufen werden, bis der Rechner eine Betätigung der Tastatur registriert hat. Dies ist genau dann der Fall, wenn die Variable Y$ eine Information enthält:

```
20 IF Y$="" THEN GOTO 10
30 REM VERARBEITUNG DER INFORMATION IN Y$
```

In Zeile 30 beginnt nun eine Verarbeitung der eingelesenen Information, d. h. des eingelesenen Zeichens. Zuerst wird dieses auf dem Bildschirm ausgegeben

```
40 PRINT Y$;
```

um dann als zuletzt eingegebenes Zeichen an das Ende einer Zeichenkette gehängt zu werden, die alle schon bisher eingegebenen Zeichen erfaßt hat:

```
50 IN$ = IN$ + Y$
60 GOTO 10
```

Verdeutlichen Sie sich dies noch einmal an folgendem Ablaufplan:

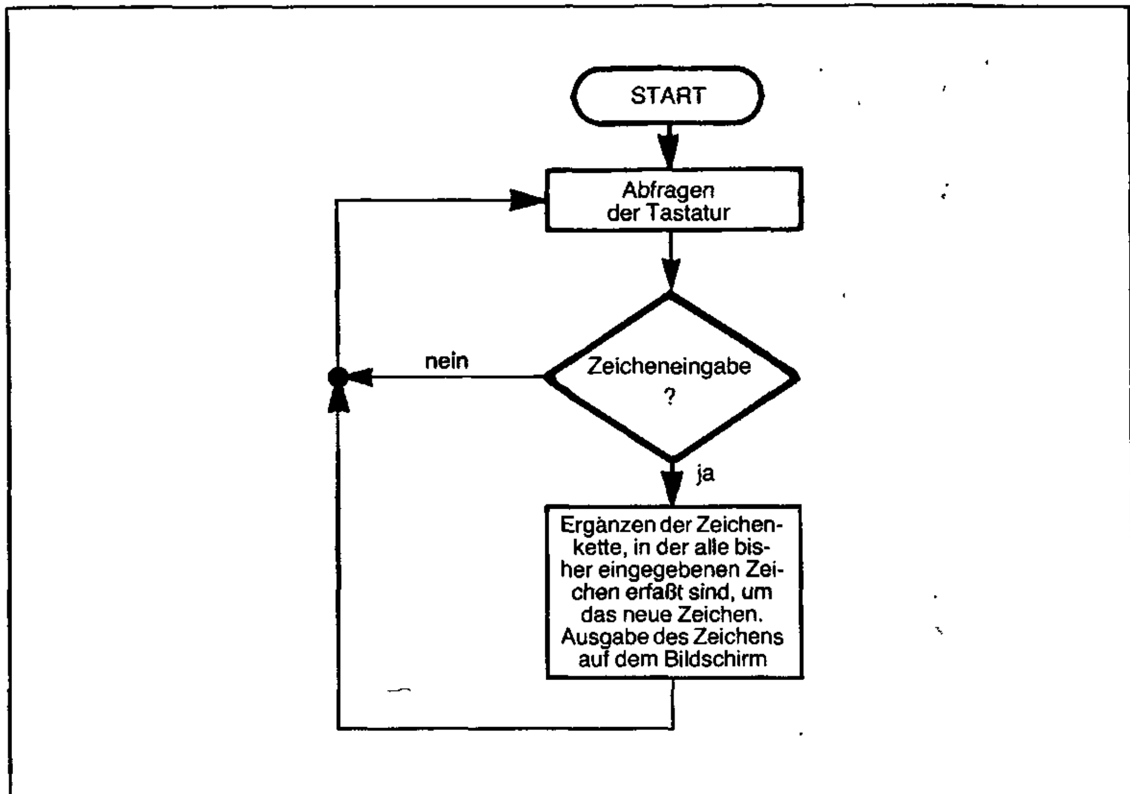


Abb. 4.2: Ablaufplan einer Tastaturabfrage

Versuchen Sie nun anhand dieses Ablaufplans Vorteile und Schwachstellen dieses Systems zu erkennen:

Vorteile

* Es werden alle Zeichen, die von der Tastatur eingegeben werden, bearbeitet und einer Zeichenkette zugewiesen.
* Die Übergabe eines eingegebenen Zeichens an die Variable erfolgt sofort nach Betätigung einer Taste, nicht, wie bei der INPUT-Anweisung, erst nach Drücken der RETURN-Taste.

* Die Länge der für die Eingabe zulässigen Zeichen kann begrenzt werden (z. B. durch IF LEN(IN$)=5 THEN GOTO XXX).
* Bestimmte Zeichen können als unzulässig deklariert werden (z. B. IF Y$="0" THEN GOTO 10).

Nachteile

* Es ist kein Cursor auf dem Bildschirm zu sehen.
* Es fehlt eine Programmzeile, in der dem Rechner mitgeteilt wird, daß bei Eingabe eines bestimmten Zeichens (z. B. RETURN) der Einlesevorgang beendet ist.
* Auch Steuerzeichen, wie z. B. der Linkspfeil, werden an die Zeichenkette übergeben.

Wir wollen nun Schritt für Schritt anhand des Ablaufplans die Nachteile einer INKEY$-Eingaberoutine beheben und ein paar der Vorzüge einbauen.

Als erster Nachteil erwies sich, daß kein Cursor auf dem Bildschirm zu sehen war. Dafür gibt es ein Steuerzeichen, das Sie auf Seite 145 Ihres Handbuchs finden. Es heißt „Cursor ein" und hat den ASCII-Code 14. Geben Sie NEW ein, und testen Sie anhand des Programms in Abb. 4.4, wie Sie mit Hilfe einer PRINT-Anweisung einen blinkenden Cursor auf dem Bildschirm erzeugen:

```
10 CLS
20 PRINT CHR$(14);
30 GOTO 30
```

Was nun fehlt, ist eine Tastaturabfrage. Dafür eignet sich Programmzeile 30:

```
30 Y$=INKEY$ :IFY$<>""THEN GOTO 100
```

Der nächste Schritt ist nun, eine Programmzeile 100 zu erstellen.

```
100 REM ZU DIESER ZEILE WIRD VERZWEIGT, WENN EIN
         ZEICHEN AUF DER TASTATUR EINGEGEBEN WURDE.
```

Hinter der Zeile 100 erfolgt nun die Bearbeitung des eingegebenen Zeichens. Nachteil 1 bei der Verwendung von INKEY$-Routinen, ein fehlender Cursor, ist an dieser Stelle schon behoben.

Der nächste Nachteil in der Liste ist der, daß der Rechner bei Verwendung einer INKEY$-Routine von sich aus nicht in der Lage ist, zu erkennen, wann eine Eingabe abgeschlossen ist.

Eine Eingabe soll in der Regel dann zu Ende sein, wenn

a) die RETURN-Taste betätigt wird oder
b) die Länge der Eingabe auf eine bestimmte Anzahl von Zeichen begrenzt ist und diese bereits eingegeben wurden.

Wenn Sie also in einem Programm z. B. die Eingabe einer Postleitzahl wünschen, so ist diese als erfolgt zu betrachten, wenn Sie 4 Ziffern eingegeben haben, da es in Deutschland keine 5- oder 6-stelligen Postleitzahlen gibt. Genauso müssen Sie aber auch die Möglichkeit schaffen, die Eingabe auch schon früher beenden zu können, da die Eingabe einer 8 für München oder 43 für Essen als Postleitzahl vollkommen ausreichend ist.

Um dies zu bewerkstelligen, müssen Sie das Programm mit einem Zeichenzähler ZZ versehen, der jedesmal, wenn der Variablen Y$ in der INKEY$-Routine ein Wert übergeben wurde, aktualisiert wird.

Dieses „Aktualisieren“ muß so aussehen, daß der Zähler jeweils um 1 erhöht wird, wenn es sich im Sinne der Eingabe um ein zulässiges Zeichen handelt, d. h. wenn Buchstaben, Zahlen oder Sonderzeichen eingegeben werden. Bei Eingabe von Steuer- bzw. Korrekturzeichen, wie z. B. dem Pfeil nach links (Backspace, letztes Zeichen löschen), muß der Zähler entsprechend vermindert werden. Im ersteren Falle wird das Zeichen in Y$ jeweils an die Zeichenkette, in der alle bisher eingegebenen Zeichen erfaßt sind, angehängt:

IN$ = IN$ + Y$

Im letzten Falle (Korrektur) muß das letzte Zeichen aus der Zeichenkette IN entfernt werden:

IN$ = LEFT$(IN$,LEN(IN$)−1)

Durch die LEFT$-Anweisung wird, wie Sie Ihrem Handbuch auf Seite 90 entnehmen können, eine Zeichenkette wie IN$ „rechts abgeschnitten“.

Nehmen wir an, IN enthielte vor Aufruf der LEFT$-Funktion die Zeichenkette "12345“. Das sind 5 Zeichen. Somit ist LEN(IN$)=5. Nun wird in der o. a. Anweisung der Variablen IN$ eine Zeichenkette zugewiesen, die LEN(IN$)−1 Zeichen lang sein soll. Wenn z. B.

IN$="12345“ ist und

LEFT$(IN$,LEN(IN$)−1) (LEN(IN$)−1 entspricht 5−1 = 4)

erzeugt werden soll, erhält man als Ergebnis die ersten 4 Zeichen aus IN$, also "1234“. Durch

IN$=LEFT$(IN$,LEN(IN$)−1)

wird also aus dieser Zeichenkette jeweils das letzte Zeichen entfernt. Dies wollen wir nun in unerem Ablaufdiagramm in Abb. 4.3 berücksichtigen:

Übergeben wird die maximale Wortlänge WL = n Zeichen.

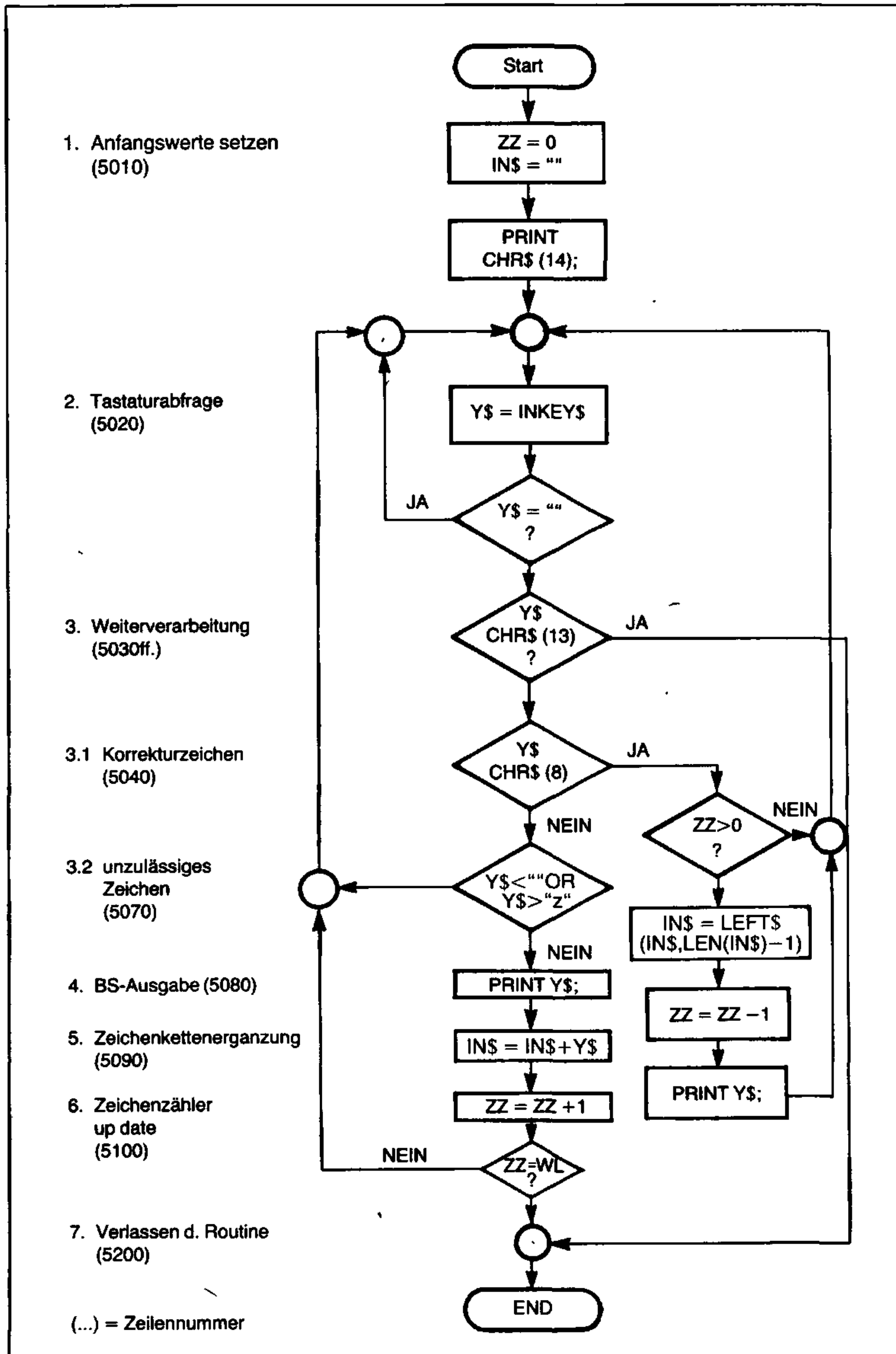


Abb. 4.3: Ablaufdiagramm einer Tastaturabfrageroutine

```
5000  'TASTATUREINGABEROUTINE ***********
5010  ZZ=0:IN$="":PRINTCHR$(14);
5020  Y$=INKEY$:IFY$=""THEN5020
5030  IFY$=CHR$(13)THEN 5200
5040  IFY$<>CHR$(8)THEN5070
5050  IFZZ>0THEN IN$=LEFT$(IN$,LEN(IN$)-1):ZZ=ZZ-1
      :PRINTY$;
5060  GOTO 5020
5070  IF Y$<" " OR Y$>"z" THEN 5020
5080  PRINTY$;
5090  IN$=IN$+Y$
5100  ZZ=ZZ+1
5110  IF ZZ=WL THEN 5200
5120  GOTO 5020
5200  RETURN
5210  '************************************
```

Abb. 4.4: Das dazugehörige Programm

Um diese Anweisungsfolge mehrmals in einem Programm verwenden zu können, empfiehlt es sich, sie in ein Unterprogramm zu fassen. Immer, wenn eine Eingabe erfolgen soll, wird durch ein

```
GOSUB 5000
```

zu der Unterroutine verzweigt. Übergeben wird über die Variable WL die Anzahl der maximal für die Eingabe zulässigen Zeichen. Wie Sie dem Beispiel in Abb. 4.5 entnehmen können, sind dies 4 Zeichen für die Postleitzahl. Wenn Sie nun versuchen, eine 5-stellige Postleitzahl einzugeben, „streikt" der Rechner. Angenehmerweise tut er dies jedoch, ohne eine Fehlermeldung auszugeben: Der Rechner nimmt weitere Zeichen einfach nicht an.

Es ist uns also gelungen, eine Eingabe auf eine bestimmte Länge zu begrenzen und die Eingabe „unerwünschter Zeichen", wie Pfeil hoch, Pfeil rechts etc. durch Programmzeile 5070 der Routine zu unterbinden:

```
5070 IF Y$<" " OR Y$>"z" THEN 5020
```

Schauen Sie in der ASCII-Tabelle auf Seite 145 Ihres Handbuchs nach, und überprüfen Sie, welche Zeichen hier bei der Eingabe vom Rechner ignoriert werden. Es sind – vielleicht unabsichtlich auf der Tastatur betätigte – Steuerzeichen wie die oben erwähnten und Grafikzeichen. Beide haben in einer Postleitzahl nichts zu suchen.

Bei anderen Eingaben mag es Ihnen wünschenswert erscheinen, gerade Grafikzeichen oder z. B. nur bestimmte Grafikzeichen für die Eingabe zuzulassen:

```
5070 IF Y$<CHR$(128) OR Y$>CHR$(191) THEN 5020
```

```
10    PRINT"WIE LAUTET DIE POSTLEITZAHL :";
20    WL=4:GOSUB 5000
30    IN=VAL(IN$)
40    PRINT"OK. POSTLEITZAHL ";IN
50    END
60    '


5000  'TASTATUREINGABEROUTINE ***********
5010  ZZ=0:IN$="":PRINTCHR$(14);
5020  Y$=INKEY$:IFY$=""THEN5020
5030  IFY$=CHR$(13)THEN 5200
5040  IFY$<>CHR$(8)THEN5070
5050  IFZZ>0THEN IN$=LEFT$(IN$,LEN(IN$)-1):ZZ=ZZ-1
      :PRINTY$;
5060  GOTO 5020
5070  IF Y$<" " OR Y$>"z" THEN 5020
5080  PRINTY$;
5090  IN$=IN$+Y$
5100  ZZ=ZZ+1
5110  IF ZZ=WL THEN 5200
5120  GOTO 5020
5200  RETURN
5210  '***********************************
```

Abb. 4.5: Einlesen von Werten mit der Eingaberoutine

Genauso ist es auch möglich, nur Zahlen oder nur große Buchstaben für die Eingabe zuzulassen:

```
5070 IF Y$<"0" OR Y$>"9" THEN 5020
```

oder

```
5070 IF Y$<"A" OR Y$>"Z" THEN 5020
```

Das Problem, das sich hier bei häufiger Verwendung der Eingaberoutine im gleichen Programm stellt, ist, daß Sie sich für eine bestimmte Überprüfung in der Zeilennummer 5070 entscheiden müssen und die derart fixierten Eingabebeschränkungen für das ganze Programm verbindlich sind. Dazu später mehr.

Der Grund, weshalb man solche Abfragen in eine Eingaberoutine mit einbaut, ist am besten mit einem Sprichwort zu beantworten: Ein Übel bekämpft man an der Wurzel. Stellen Sie sich vor, wie kompliziert sich die (nachträgliche) Überprüfung einer Zeichenkette gestaltet, die nur Hexadezimalwerte enthalten soll, also die Ziffern 0–9 und A–F. In der Eingaberoutine ist das eine Zeile:

```
5070 IF Y$>"F" OR (Y$<"A"AND NOT (Y$>="0" AND Y$<"9"))
THEN 5020
```

In einer Zeichenkette muß jedes einzelne Zeichen (nachträglich) in einer Schleife auf diese Bedingung hin geprüft werden. Das bedeutet überflüssige Vergeudung von Speicherplatz und Rechenzeit. Geschieht eine solche Abfrage nicht, riskieren Sie das Auftreten eines Fehlers. Es ist dann nicht auszuschließen, daß Ihr Programm nicht mehr läuft oder falsche Ergebnisse liefert.

Je komfortabler eine solche Eingaberoutine aufgebaut ist, desto mehr „Arbeit" kann Sie dem Programmierer abnehmen. Komfortabel bedeutet jedoch auch komplex, denn diese Unterroutine müßte um weitere Bestandteile erweitert werden. Dies ist jedoch nur dann sinnvoll, wenn ein Programm viele verschiedenartige Eingaben erfordert oder z. B. mit sog. Eingabemasken gearbeitet wird. Eine Eingabemaske ist ein „Bild" auf dem Bildschirm, wo die Stellen, an denen Eingaben gemacht werden sollen, ausgepunktet sind, wie z. B. in folgender Bildschirmmaske:

```
<@> = EINGABE WIEDERHOLEN

Vorname  : ■..............
Nachname : ...............
PLZ      : ....
ORT      : ...............
TELEFON:
Vorwahl  : ......
Nummer   : .......
```

Abb. 4.6: Bildschirmmaske

Eine Eingaberoutine, die Eingaben über derartige Bildschirmmasken unterstützt, finden Sie in Abb. 4.7. Ihr detaillierter Aufbau soll hier nicht beschrieben werden. Sie ist jedoch den bisher beschriebenen Eingaberoutinen ähnlich. Ihre zusätzlichen Fähigkeiten bestehen insbesondere darin, daß

1. über die Variable FL gekennzeichnet wird, welche Zeichen im Sinne

der Eingabe zulässig sind. Ist FL negativ, können nur Ziffern und Vorzeichen eingegeben werden. Ist FL positiv, werden auch Zeichenketten vom Rechner akzeptiert. Der Wert, der in WL vorgegeben wird, kennzeichnet nach wie vor die maximale Eingabelänge.

2. Wird das Zeichen <@> betätigt, setzt der Rechner die Variable KF, die sonst immer den Wert 0 besitzt, auf 1 und unterbricht die gegenwärtige Eingabe.

   Schauen Sie sich die Bildschirmausgabe in Abb. 4.8 an. Hinter dem Text „Vorname:“, der durch

   40 PRINT "Vorname: ....";

ausgegeben wird, stehen 15 Punkte, die die Maximallänge der Eingabe kennzeichnen:

120 PRINT @ 131,;:FL=15:GOSUB 230: ...

An dieser Stelle erfolgt die Verzweigung zur Unterroutine. Durch FL=15 werden maximal 15 Zeichen (Buchstaben etc.) verarbeitet und der Variablen IN$ zugewiesen. Stünde in Zeile 120 FL=−15, so könnten nur Zahlen eingegeben werden. Dies ist jedoch für die Eingabe eines Vornamens unsinnig. Bei der Eingabe der Postleitzahl (im Programm Abb. 4.8 Zeile 140) erweist sich auch diese Fähigkeit der Unterroutine als zweckmäßig.

In jeder Zeile wird nun durch

... IF CF=1 THEN GOTO ...

im Falle einer Betätigung der <@>-Taste eine erneute Eingabe eingeleitet und die gegenwärtige unterbrochen. Verzweigt der Rechner zur Folgezeile, ist eine Eingabe erfolgt.

Auf diese Weise wird eine komplette Bildschirmmaske versorgt.

```
6000  'Uebergabeparameter:
6001  'FL=MAXIMALLAENGE DER ZEICHENKETTE.
6002  '  Ist FL negativ, koennen nur
6003  '  Zahlen, keine Buchstaben oder
6004  '  Sonderzeichen eingegeben werden.
6005  '
6006  'Ruecklaufparameter:
6007  'CF=1 DIE EINGABE WURDE UNTERBROCHEN.
6008  '  CF=0 Die Eingabe hat Gueltigkeit.
6009  '  WD=0 Der eingegebene Zahlenwert ist
6010  '       ganzzahlig.
```

Abb. 4.7: Eine Tastatureingaberoutine, die auch Bildschirmmasken unterstützt

```
6011  '  WD=1 Der eingegebene Zahlenwert ist
6012  '       nicht ganzzahlig.
6013  '  WS=0 Es wurde eine positive Zahl
6014  '       eingegeben.
6015  '  WS=1 Es wurde eine negative Zahl
6016  '       eingegeben.
6017  ' Die eingelesenen Werte stehen immer in der
       Variablen
6018  '  IN$
6019  'Beispielaufruf:
6020  ' 10 FL=-4:GOSUB8000:IF CF=1 THEN
6021  '    PRINT@200,"Eingabekorrektur";:GOTO 10
6031  '    ELSE WERT$=IN$
6041  '
6051  '
6061  '
7998  STOP
7999  'HIER BEGINNT DIE EINGABEROUTINE
8000  CF=0:PRINTCHR$(14);:IN$="":Y$=INKEY$:WD=0:WS
      =WD:WL=WD:IFFL=WDTHENFL=1
8001  PRINTSTRING$(ABS(FL),".");STRING$(ABS(FL),24
      );
8002  Y$=INKEY$:IFY$=""THEN8002
8003  IFY$<>CHR$(13)THEN8005ELSEPRINTSTRING$(ABS(F
      L)-WL," ");
8004  RETURN
8005  IF Y$<>"@"THEN8007
8006  CF=1:RETURN
8007  IFY$=CHR$(24)THENPRINTSTRING$(WL,CHR$(24));:
      GOTO8000
8008  IF Y$<>CHR$(8)THEN8012 ELSE IF WL=0THEN 8002
       ELSEPRINTCHR$(8);STRING$(ABS(FL)-WL,".");ST
      RING$(ABS(FL)-WL,24);:IFFL>0THEN8010 ELSE GO
      TO 8011
8010  IN$=LEFT$(IN$,LEN(IN$)-1)
8011  WL=WL-1:GOTO8002
8012  IF ABS(FL)=WL THEN 8002 ELSE IF FL>0 THEN IF
       Y$>=" "AND Y$<"z"THEN8017
8013  IF Y$="."ANDWD=0THENWD=1:GOTO8017
8014  IF Y$=","THENPRINTY$;:WL=WL+1:GOTO8018
8015  IF (Y$="-"ORY$="+")ANDWS=0ANDWL=0THENWS=1:GO
      TO8017
8016  IFY$<"0" OR Y$>"9"THEN8002
8017  PRINTY$;:IN$=IN$+Y$:WL=WL+1
8018  IF ABS(FL)=1 THEN 8004 ELSE 8002
8019  ' ENDE DER EINGABEROUTINE
```

Abb. 4.7: Eine Tastatureingaberoutine, die auch Bildschirmmasken unterstützt (Fortsetzung)

```
1       Anwendungsbeispiel Bildschirmmaske
10     CLEAR 2000
20     CLS:PRINT"<@> = EINGABE WIEDERHOLEN"
30     PRINT:PRINT
40     PRINT"Vorname  : ..............."
50     PRINT"Nachname : ..............."
60     PRINT"PLZ      : ...."
70     PRINT    "ORT      : ..............."
80     PRINT"TELEFON:"
90     PRINT"Vorwahl  : ......"
100    PRINT"Nummer   : ......."
110    'ENDE DER MASKE
120    PRINT@131,;:FL=15:GOSUB 230: IF CF=1 THEN 12
       0 ELSE VN$=IN$
130    PRINT@171,;:FL=15:GOSUB230:IF CF=1 THEN 130
       ELSE NN$=IN$
140    PRINT@211,;:FL=-4:GOSUB 230: IF CF=1 THEN 14
       0 ELSE PL$=IN$
150    PRINT@251,;:FL=15:GOSUB230:IF CF=1 THEN 150
       ELSE O$=IN$
160    PRINT@331,;:FL=-6:GOSUB 230:IF CF=1 THEN 160
        ELSE VW$=IN$
170    PRINT@371,;:FL=-7:GOSUB230:IFCF=1 THEN170 EL
       SE RN$=IN$
180    PRINT:PRINT:PRINT
190    'ENDE DER EINGABEN
200    PRINT"FOLGENDE DATEN WURDEN EINGELESEN:"
210    PRINTVN$:PRINTNN$:PRINTPL$,O$:PRINTVW$,RN$
220    END
230    GOTO 8000
7998   STOP
7999   'HIER BEGINNT DIE EINGABEROUTINE
8000   CF=0:PRINTCHR$(14);:IN$="":Y$=INKEY$:WD=0:WS
       =WD:WL=WD:IFFL=WDTHENFL=1
8001   PRINTSTRING$(ABS(FL),".");STRING$(ABS(FL),24
       );
8002   Y$=INKEY$:IFY$=""THEN8002
8003   IFY$<>CHR$(13)THEN8005ELSEPRINTSTRING$(ABS(F
       L)-WL," ");
8004   RETURN
8005   IF Y$<>"@"THEN8007
8006   CF=1:RETURN
8007   IFY$=CHR$(24)THENPRINTSTRING$(WL,CHR$(24));:
       GOTO8000
8008   IF Y$<>CHR$(8)THEN8012 ELSE IF WL=0THEN 8002
        ELSEPRINTCHR$(8);STRING$(ABS(FL)-WL,".");ST
       RING$(ABS(FL)-WL,24);:IFFL>0THEN8010 ELSE GO
       TO 8011
8010   IN$=LEFT$(IN$,LEN(IN$)-1)
8011   WL=WL-1:GOTO8002
8012   IF ABS(FL)=WL THEN 8002 ELSE IF FL>0 THEN IF
        Y$>=" "AND Y$<"z"THEN8017
8013   IF Y$="."ANDWD=0THENWD=1:GOTO8017
8014   IF Y$=","THENPRINTY$;:WL=WL+1:GOTO8018
8015   IF (Y$="-"ORY$="+")ANDWS=0ANDWL=0THENWS=1:GO
       TO8017
8016   IFY$<"0" OR Y$>"9"THEN8002
8017   PRINTY$;:IN$=IN$+Y$:WL=WL+1
8018   IF ABS(FL)=1 THEN 8004 ELSE 8002
8019   ' ENDE DER EINGABEROUTINE
```

Abb. 4.8: Ein Anwendungsbeispiel mit Bildschirmmasken

Kapitel 5

Tonerzeugung mit dem Colour-Genie

5.1 DIE BEFEHLE PLAY UND SOUND

In Ihrem Colour-Genie befindet sich ein Soundgenerator. Ein Soundgenerator ist ein Chip, mit dem Töne erzeugt werden können. Im Microsoft-BASIC stehen Ihnen zwei Anweisungen zur Verfügung, um diesen Baustein anzusprechen: PLAY und SOUND.

Mit PLAY wird ein Ton erzeugt, mit SOUND werden dessen Eigenschaften verändert:

```
PLAY (1,3,1,15)
```

Wenn Sie nun am Lautstärkeregler Ihres Fernsehers drehen, hören Sie einen Ton, der der Note C entspricht.

(Sie können auch, um ein sauberes Tonsignal zu erhalten, über die Audio-Buchse an der Rückwand des Colour-Genie einen Verstärker anschließen.) Ein einmal programmierter Ton wird so lange ausgegeben, bis dem Soundgenerator eine Anweisung erteilt wird, diesen wieder abzustellen:

```
PLAY (1,3,1,0)
```

Die Bedeutung der einzelnen PLAY-Parameter können Sie auf Seite 111 Ihres BASIC-Handbuchs nachlesen. Sie sehen, daß der Soundgenerator recht vielseitig einsetzbar ist. Um sich die Tonleiter in C-Dur ausgeben zu lassen, geben Sie folgendes Programm ein:

```
10 'DIE C-DUR TONLEITER VOM TIEFSTEN
BIS ZUM HOECHSTEN TON
20 FOR O=1 TO 8 :'ALLE OKTAVEN
30  FOR T=1 TO 7 :'CDEFGAB
40   PLAY(1,O,T,15)
50   FOR D=1 TO 10:NEXT D
60  NEXT T
70 NEXT O
```

Abb. 5.1: Die C-Dur Tonleiter

Dies ist die verfügbare Tonpalette, die hier über Kanal 1 des Tonbausteins ausgegeben wird. Der Soundgenerator verfügt über drei solcher Kanäle. Jeder kann getrennt angesprochen werden, mit verschiedenen Tönen, die dann, zusammen ausgegeben, z. B. einen Akkord liefern können:

```
10 'EIN AKKORD WIRD GESPIELT
20 PLAY (1,4,1,15):'C AUF KANAL 1
30 PLAY (2,4,3,15):'E AUF KANAL 2
40 PLAY (3,4,5,15):'G AUF KANAL.3
```

Abb. 5.2: Spielen von Akkorden

Jeden ausgegebenen Ton können Sie durch Eingabe von

SOUND 7,63

wieder abstellen. Töne oder Akkorde können Sie auch langsam ausklingen lassen, indem Sie die Lautstärke dezimieren:

```
10 FOR L=15 TO 0 STEP -1
20 PLAY (1,4,1,L):'C AUF KANAL 1
30 PLAY (2,4,3,L):'E AUF KANAL 2
40 PLAY (3,4,5,L):'G AUF KANAL 3
50 FOR D=1 TO 10:NEXT D:'VERZOEGERUNG
60 NEXT. L
```

Abb. 5.3: Dezimierung der Lautstärke

Mit dem Soundgenerator lassen sich jedoch nicht nur Töne erzeugen. Auch sogenannte Effekte können in die Tonausgabe miteingebaut werden. So kann z. B. ein Ton nach bestimmten Vorgaben derart moduliert werden, daß sich Echohalleffekte erzeugen lassen oder daß im Hintergrund ein leichtes Meeresrauschen mitklingt. Wenn Sie sich mit solchen Dingen näher befassen wollen, müssen Sie sich das technische Handbuch zum Colour-Genie zur Hand nehmen. Dort ist auf Seite 21ff. der interne Aufbau des Soundgenerators beschrieben. Für den Fall, daß diese Unterlagen nicht ausreichen sollten, können Sie bei der Herstellerfirma des Soundgenerators ein

USER-MANUAL zum Soundgeneratorbaustein AY-3-8910

anfordern. Dort sind alle Eigenschaften des Bausteins und dessen Programmierung beschrieben. Zum Testen der Effektroutinen steht Ihnen im Anhang des BASIC-Manuals ein Programm zur Verfügung.

5.2 EIN MUSIKINTERPRETER

Wenn Sie einmal angefangen haben, ein Musikinstrument zu erlernen, haben Sie bestimmt irgendwo einige Notenblätter von Musikstücken. In Abb. 3.28 wird Ihnen ein Programm vorgestellt, mit dessen Hilfe Sie direkt in solchen Kompositionen Noten in den Rechner eingeben können, die dann vom Soundgenerator Ihres Colour-Genie gespielt werden. Die Befehle, die dieser Noteninterpreter kennt, sind recht einfach aufgebaut:

Zuerst einmal gibt es die Noten C, D, E, F, G, A, B und die Pause P. Hinter einer Note kann (muß nicht) eine Oktave angegeben werden, in der die entsprechende Note gespielt werden soll, wie z. B.:

C3,D5,E7

Wird hinter der Note kein Zahlenwert vorgegeben, entspricht dies der Oktave 4. Wenn Sie also

C,D,E,F,G

eingeben, werden all diese Noten in der vierten Oktave gespielt.

Um Halbtöne anzusprechen (entsprechend den schwarzen Tasten auf der Klaviatur, s. a. S. 111), wird an die entsprechende Note ein .#. angehängt. Beispiel:

C3# spielt die Note cis in der 3. Oktave.

Nun zur Tonlänge. Alle bisher behandelten Eingaben erzeugen Töne, die genau einen Takt lang sind. Um halbe Noten, Viertel, Achtel, Sechzehntel etc. zu spielen, wird an die Note jeweils ein / und eine Zahl angehängt:

C/2 entspricht einer Note von halber Tonlänge;
C/4 entspricht einer Viertel-Note;
C/8 einer Achtel-Note etc.

Manchmal tauchen in Kompositionen sog. punktierte Noten auf. Eine punktierte halbe Note hat z. B. 3/4 Taktlänge. Auch die punktierten Noten wurden im Musikinterpreter in Abb. 5.4 berücksichtigt. Der Punkt wird einfach an die entsprechende Note angehängt, wie z. B.

C/4.

Soweit zur Syntax des Musikinterpreters. Alle Noten werden, durch Kommata oder Leerzeichen voneinander getrennt, in Variablen, beginnend mit A$(0) ab Programmzeile 30 abgelegt. Die Noten werden so lange abgearbeitet, bis der Rechner ein A$(x) vorfindet, das einen Leerstring enthält. Alle Daten für den Soundgenerator werden zuerst im Rechnerspeicher ab Speicheradresse F400 Hex ff. abgelegt. Erst wenn alle Noten in den A$-Variablen analysiert wurden, wird das Musikstück gespielt.

Bei falschen Eingaben werden vom Interpreter Fehlermeldungen ausgegeben. Der Interpreter kennt 2 Fehlermeldungen:

a) Unbekanntes Zeichen
b) Unzulässig benutztes Zeichen

Die erste Fehlermeldung wird immer dann ausgegeben, wenn sich in der Variablen A(x) ein unzulässiges Zeichen befindet. Über die zweite Fehlermeldung werden unzulässige Noten (solche, die es nicht gibt) angezeigt: E#, H#.

```
10    CLEAR 2000:DIM A$(20):SP=&HF400:PP=SP
20    ' AB HIER DIE NOTEN ABLEGEN
30    A$(0)="G/4,E/4,E/2,F/4,D/4,D/2,C/4,D/4,E/4,F
      /4,G/4,G/4,G/2,"
40    '
50    '
60    '
70    '
80    '
999   '
1000  CLS:PRINT"UEBERSETZUNG BEGINNT"
1010  O=4:'VOREINSTELLUNG OKTAVE
1020  P=1:' 1.ZEICHEN
1030  FOR X=0 TO 20
1035  PRINTSTRING$(30,"*"):PRINTA$(X):PRINTSTRING$
      (30,"*"):PRINT
1040  L=LEN(A$(X)):IF L=0 THEN 10000
1045  W1=64:W2=1:W3=O:W5=15
1050  IF P>L THEN P=1:NEXTX ELSE B$=MID$(A$(X),P,1
      ):B=ASC(A$(X))
1060  S$=" ,CDEFGAB/#.P":FOR U=1 TO 13
1080  S1$=MID$(S$,U,1):IF S1$=B$ THEN 1200
1090  NEXTU
1100  PRINT"FEHLER IN A$(";X;"):UNBEK. ZEICHEN"
1110  PRINTA$(X)
1120  PRINTTAB(P-1)"!"
1130  STOP
1200  ON U GOTO 1300,1300,1400,1500,1600,1700,1800
      ,1900,2000,2100,2200,2300,2400
1201  GOTO 1100
1300  ' ---->  ,
1301  'PLAY
1302  PRINT"NOTENLAENGE ";W1
1303  PRINT"OKTAVE      ";W3
1304  PRINT"NOTE        ";W4
1305  PRINT"LAUTSTAERKE ";W5
1306  POKE PP,W1AND255:PP=PP+1:POKEPP,INT(W1/256):
      PP=PP+1:POKEPP,W3:PP=PP+1:POKEPP,W4:PP=PP+1:
      POKEPP,W5:PP=PP+1
1307  PRINT
1308  P=P+1:GOTO 1045
1400  ' ----> C
1410  W4=1:GOSUB9000:GOTO1050
```

Abb. 5.4: Musikinterpreter

```
1500  ' ----> D
1510  W4=2:GOSUB9000:GOTO1050
1600  ' ----> E
1610  W4=3:GOSUB9000:GOTO1050
1700  ' ----> F
1710  W4=4:GOSUB9000:GOTO1050
1800  ' ----> G
1810  W4=5:GOSUB9000:GOTO1050
1900  ' ----> A
1910  W4=6:GOSUB9000:GOTO1050
2000  ' ----> B
2010  W4=7:GOSUB 9000:GOTO1050
2020  '
2100  ' ----> /
2110  C1$=""
2120  P=P+1:B$=MID$(A$(X),P,1)
2130  IF (B$<>",") AND (B$<>" ") AND (B$<>".") THE
      N C1$=C1$+B$:GOTO 2120
2140  W1=W1/VAL(C1$):GOTO1050
2200  ' ----> #
2210  IF (W4=1) OR (W4=2) THEN W4=W4+7:P=P+1:GOTO1
      050
2220  IF (W4=3) OR (W4=7) THEN PRINT"FEHLER IN A$(
      ";X;"):UNZULAESSIG BENUTZTES ZEICHEN":GOTO11
      10
2300  ' ----> .
2310  W1=W1+.5*W1:P=P+1:GOTO1050
2400  ' ----> P
2410  W5=0:P=P+1:GOTO1050
2420  '

9000  ' UNTERPROGRAMM:
9010  'CHECKEN OB NOTE + OKTAVENWERT
9020  P=P+1
9030  B$=MID$(A$(X),P,1):IF B$<"1" OR B$>"8" THEN
      RETURN
9040  W3=VAL(B$):P=P+1:RETURN
9050  '

10000 'ABSPIELEN DER NOTEN
10010 PRINT"<RETURN> ZUM ABSPIELEN"
10020 PRINTINT((PP-SP)/5);" NOTEN SIND ZU SPIELEN"

10030 '
10040 FOR X=SP TO PP-4 STEP 5
10050 P1=PEEK(X)+256*PEEK(X+1)   :'TONLAENGE
10060 P2=PEEK(X+2):'OKTAVENNUMMER
10070 P3=PEEK(X+3):'TONNUMMER
10080 P4=PEEK(X+4):'LAUT:0=PAUSE 15=TON
10090 '
10099 PRINT"PLAY(1,";P2;",";P3;",";P4;")"
10100 FOR L=1 TO P1
10111 I=ADATA (1,P2,P3,P4)
10120 NEXT L
10130 I=ADATA (1,P2,P3,0):'ENDE TON
10140 NEXT X
10145 I=ADATA(1,P2,P3,0)
10150 END
```

Abb. 5.4: Musikinterpreter (Fortsetzung)

Kapitel 6

Arbeiten mit Disk-BASIC

Wenn Sie eine Diskettenstation für das Colour-Genie erwerben, bekommen Sie automatisch das Disk-BASIC mitgeliefert. Es befindet sich auf dem Steckmodul, das rechts hinten am Rechner eingesteckt wird. Dieses Disk-BASIC enthält die notwendigen Programme, um mit den Diskettenlaufwerken unter Microsoft-BASIC arbeiten zu können. Beim Einschalten eines Rechners, in den der Disk-BASIC Einschub eingesteckt ist, stehen alle notwendigen Routinen automatisch zur Verfügung. Dies ist jedoch nur eines der kennzeichnenden Merkmale des Disk-BASIC. Auch solche Anweisungen, die nicht direkt mit dem Peripheriegerät Diskette zu tun haben, stehen neu, sozusagen als erweiterter Befehlssatz des Interpreters, zur Nutzung bereit. Dazu zählen

Anweisungen wie LINEINPUT
Funktionen wie INSTR,MID (....)=, DEF FN
Echtzeituhr und deren Programmierung

Einige dieser neuen Möglichkeiten sollen Sie im folgenden kennenlernen.

6.1 DIE ZEIT

Eine interessante Funktion ist TIME$, die, einmal mit den richtigen Daten versehen, Datum und Uhrzeit ständig aktualisiert, solange der Rechner in Betrieb ist.

Wenn Sie

```
PRINT TIME$
```

eingeben, erhalten Sie vom Rechner eine siebzehn Zeichen lange Zeichenkette, in deren ersten acht Zeichen das Datum und in deren letzten acht Zeichen die Uhrzeit erfaßt ist. Mit Hilfe dieser Funktion ist es Ihnen möglich, bestimmte Rechenabläufe zeitlich zu erfassen oder Eingaben zeitlich zu begrenzen. Da die Werte für Tagesdatum und Uhrzeit beim Einschalten des Rechners jeweils mit 0 initialisiert werden, benötigen Sie ein kurzes Programm, das Datum und Uhrzeit mit den richtigen Werten versieht:

```
5     CLS
10    INPUT"WIE LAUTET DAS DATUM (TTMMJJ)";D$
20    INPUT"WIE LAUTET DIE UHRZEIT(STMISE)";T$
30    TAG=VAL(LEFT$(D$,2))
40    MO=VAL(MID$(D$,3,2))
50    JA=VAL(RIGHT$(D$,2))
60    POKE &H4046,TAG:POKE&H4045,MO:POKE&H4044,JA
70    ST=VAL(LEFT$(T$,2))
80    MI=VAL(MID$(T$,3,2))
90    SE=VAL(RIGHT$(T$,2))
100   POKE&H4043,ST:POKE&H4042,MI:POKE&H4041,SE
110   CLS
120   PRINT@0,TIME$
130   GOTO120
```

Abb. 6.1: Setzen der Uhr, Eingabe des Datums

Nach Einlesen der Werte wird links oben auf dem Bildschirm das Tagesdatum und die Uhrzeit angezeigt. Sie können jetzt ohne weiteres die BREAK-Taste betätigen und das Programm durch NEW löschen oder andere BASIC-Programme starten. Unabhängig davon wird jedesmal, wenn Sie

PRINT TIME$

eingeben, das aktuelle Tagesdatum und die richtige Uhrzeit angezeigt.

Eine weitere Verwendungsform der TIME$-Funktion ist die, Rechenabläufe in BASIC-Programmen zeitlich zu erfassen.

Wie viele Durchläufe und wieviel Zeit benötigt Ihr Rechner z. B., um bei zufälliger Auswahl einer Zahl zwischen 0 und 100 die Zahl 95 zu treffen?

```
10    Z=0:'ZAEHLER DER DURCHLAEUFE
20    POKE&H4041,0:POKE&H4042,0:POKE&H4043,0:'RUEC
      KSETZEN DER UHR WIE BEI EINER STOPPUHR
30    RANDOM
40    W=RND(100)
50    Z=Z+1:'HOCHZAEHLEN ZAEHLER
60    IF W=95 THEN  80:'TEXT AUSGEBEN, WENN RICHTI
      GE ZAHL GEFUNDEN WURDE
70    GOTO 30
80    T$=RIGHT$(TIME$,8):'ERFASSEN DER BENOETIGTEN
       ZEIT
90    PRINT:PRINT
100   CLS
110   PRINT"ES WAREN ";Z;" DURCHLAEUFE ERFORDERLIC
      H, UM AUS 100 ZAHLEN DIE ZAHL 95 ZU
      ERMITTELN."
120   PRINT"DER RECHNER BRAUCHTE DAZU ";MID$(T$,4,
      2);" MINUTEN UND ";RIGHT$(T$,2);" SEKUNDEN."
```

Abb. 6.2: Arbeiten mit der Uhr. Hier: Zeitliche Ermittlung von Rechenvorgängen

Eine entsprechende Bildschirmausgabe finden Sie in Abb. 6.3.

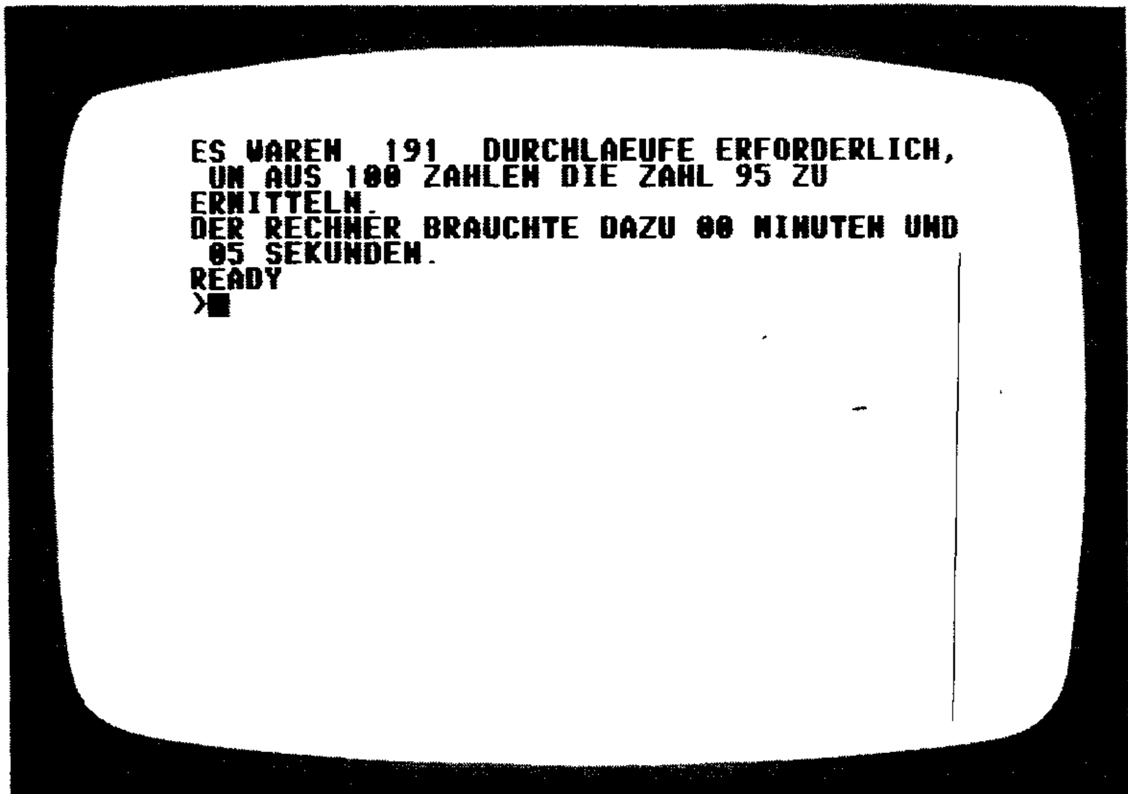


Abb. 6.3: Bildschirmausgabe: Zeitliche Ermittlung von Rechenvorgängen

Eine weitere Einsatzmöglichkeit der Echtzeituhr besteht, wie schon oben erwähnt, darin, dem Anwender für die Beantwortung einer Frage oder allgemein für die Eingabe von Zeichen nur eine begrenzte, vorgegebene Zeit zur Verfügung zu stellen. Auch dafür kurz ein Beispielprogramm:

```
10    CLS
20    PRINT"IHNEN WIRD GLEICH EINE FRAGE GESTELLT,

      ZU DER DREI MOEGLICHE ANTWORTEN VORGE-
      GEBEN WERDEN, VON DENEN NUR EINE RICHTIGIST.
      "
30    PRINT"SCHAFFEN SIE ES, DIE FRAGE IN MAXIMAL
      5 SEKUNDEN ZU BEANTWORTEN?"
40    PRINT:PRINT:PRINT"DRUECKEN SIE <RETURN> UND
      DIE ZEIT ZAEHLT"
50    INPUT A$
60    CLS
70    PRINT"WIEVIEL IST 17 MAL 16 ???"
```

Abb. 6.4: Zeitliche Begrenzung einer Eingabe

```
80    PRINT:PRINT"1 ---> 162"
90    PRINT:PRINT"2 ---> 272"
100   PRINT:PRINT"3 ---> 282"
110   PRINT:PRINT
120   POKE &H4041,0 :'SEKUNDEN AUF 0
130   Y$=INKEY$:IF RIGHT$(TIME$,2)>"05" OR Y$<>""
      THEN 140
135   GOTO 130
140   IF Y$=""THENPRINT"DIE 5 SEKUNDEN SIND UM":EN
      D
150   IFY$="2"THENPRINT"GRATULIERE! RICHTIG GELOES
      T":END
160   PRINT"DIE ANTWORT KAM ZWAR SCHNELL GENUG, AB
      ERFALSCH."
```

Abb. 6.4: Zeitliche Begrenzung einer Eingabe (Fortsetzung)

Soweit zu den Anwendungsmöglichkeiten der TIME$-Funktion.

Eine weitere unter Disk-BASIC zur Verfügung stehende Funktion, DEFFN, wird eingehend in den Kapiteln „Funktion" und „Eigene Funktionen" behandelt.

6.2 INSTR

Es gibt noch eine Funktion, die Ihnen unter Disk-BASIC zusätzlich zur Verfügung steht: Sie lautet INSTR. Mit ihrer Hilfe ist es möglich, den Inhalt von Zeichenketten dahingehend zu analysieren, ob und an welcher Stelle sich eine bestimmte Folge von Zeichen in der Zeichenkette befindet.

```
10 CLS
20 PRINT"HIER WIRD DIE HAEUFIGKEIT BESTIMM-
TER BUCHSTABEN IN EINEM TEXT ANALYSIERT"
30 LINEINPUT"WIE LAUTET DER
TEXT:";A$
40 L=LEN(A$)
50 PRINT
60 FOR X=ASC("A") TO ASC("z")
70 N=0
80 ZZ=1
90 T=0
100 N=N+1
110 ZZ=T+1
120 T=INSTR(ZZ,A$,CHR$(X))
130 IF T<>0 THEN 100
140 N=N-1
150 PRINT"<";CHR$(X);"> TAUCHT IM TEXT ";N;" MAL
AUF. DAS ENTSPRICHT EINER HAEUFIGKEIT
VON ";100*N/L;" %.":PRINT
160 NEXT X
```

Abb. 6.5: Ermittlung von Häufigkeiten mit Hilfe der INSTR-Funktion

In diesem Programm taucht neben der LINE-INPUT-Anweisung in Zeile 30, die im Gegensatz zu der INPUT-Anweisung in der Eingabe auch Kommata und ähnliche Sonderzeichen zuläßt, in Zeile 120 die INSTR-Funktion auf. Das Programm ist im wesentlichen aus zwei Schleifen aufgebaut. Die äußere Schleife beginnt in Zeile 60 und endet in Zeile 170. Sie bewirkt die Bearbeitung der Zeichen A bis z. Die innere Schleife betrifft die Zeilen 100 bis 130, wo der eingegebene Text vom ersten bis zum letzten Zeichen auf das Auffinden eines bestimmten Buchstabens hin analysiert wird. Im Programm wird mit mehreren Variablen gearbeitet, die im einzelnen folgende Bedeutung haben: Die Variable N wird jedesmal um 1 hochgezählt, wenn der Buchstabe CHR$(X) in der vorgegebenen Zeichenkette A vorgefunden wurde. Ist dies nicht der Fall, liefert die INSTR-Funktion in Zeile 20 für die Variable T den Wert Null, so daß die Anzahl der vorgefundenen Buchstaben in der Zeichenkette genau um 1 niedriger ist als N, was in Zeile 140 korrigiert wird. Die Variable ZZ stellt einen Zeiger dar, der immer auf die Stelle zeigt, bei der die Suche nach einem bestimmten Zeichen in der Zeichenkette beginnen soll. Wurde die Häufigkeit eines bestimmten Zeichens ermittelt, wird wieder zur Zeile 70 verzweigt, wo die Variablen N, ZZ und T wieder mit Anfangswerten versehen werden. Eine entsprechende Bildschirmausgabe dieses Programms finden Sie in Abb. 6.6.

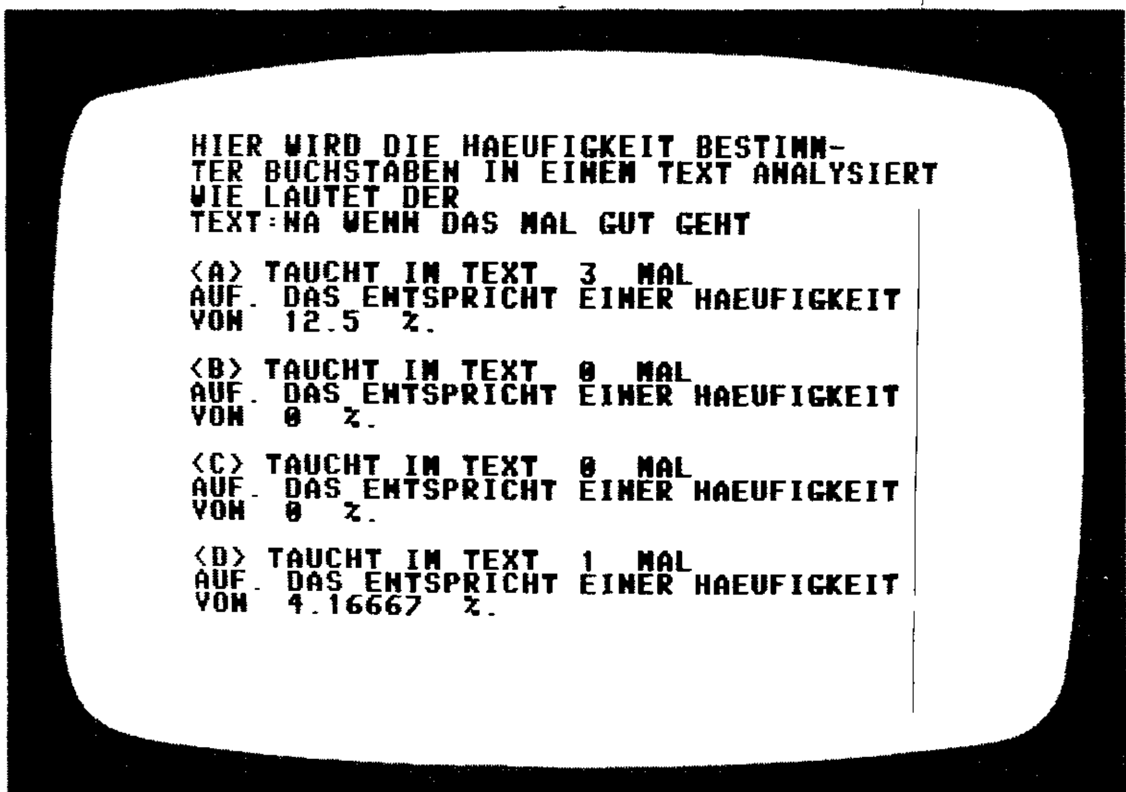


Abb. 6.6: Bildschirmausgabe im Programm in Abb. 6.5

6.3 LINE INPUT

Auch die LINE INPUT-Funktion gehört zum Bestand der Disk-BASIC-Anweisungen. Die Verwendung von LINE INPUT läßt sich schon aus dem Namen ableiten, der soviel bedeutet wie „Einlesen einer (ganzen) Eingabezeile“. Diese Anweisung wird ausschließlich zum Einlesen von Zeichenketten verwendet:

```
10 LINE INPUT"Wie war der Spielstand 1. FC Koeln gegen R.W. Essen";A$
```

Der Unterschied zur INPUT-Anweisung besteht darin, daß bei LINE INPUT keine Beschränkungen die Zuweisung bestimmter Zeichen an eine Zeichenkettenvariable verhindern.

Wenn Sie das o. a. Beispiel starten und 4:3 eingeben, wird diese Zeichenfolge auch tatsächlich der Variablen A$ zugewiesen.

Kapitel 7

Strategische Spiele

7.1 REVERSI

REVERSI ist ein orientalisches Brettspiel. Es ist auch unter dem Namen Othello geläufig. Auf einem 8 x 8 Spielfeld, wie Sie es in Abb. 7.1 sehen, werden einem Spieler runde, dem anderen viereckige Steine zugeteilt.

Es können entweder 2 Personen gegeneinander spielen oder aber eine Person gegen den Computer.

Je zwei Steine pro Spieler bilden die Ausgangsstellung.

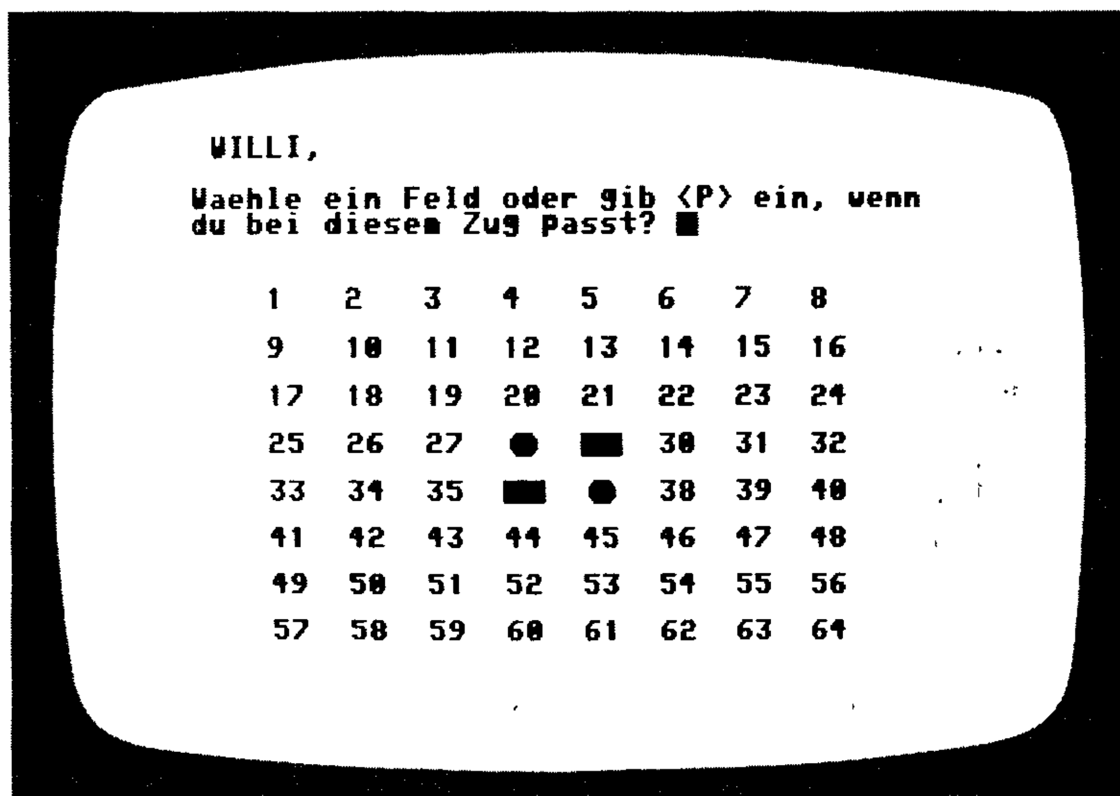


Abb. 7.1: Das durchnumerierte Spielfeld für REVERSI

Regeln:
Die beiden Spieler kommen abwechselnd zum Zuge. Der Rechner ermittelt über eine Zufallsfunktion, wer den ersten Zug hat. Mit jedem Zug wird ein neuer Stein auf das Spielfeld gesetzt. Dies geschieht durch Eingabe einer Zahl zwischen 1 und 64. Gesetzt werden darf nur auf solche Felder, die bisher noch nicht belegt sind, und zwar so, daß entweder diagonal, horizontal oder vertikal gegnerische Steine zwischen eigenen „eingeklemmt" werden. Solche „eingeklemmten" Steine werden dann zu eigenen Steinen.

Beispiel:
Wenn Sie im Spiel die eckigen Steine (xx) haben und bei folgendem Bild am Zug sind:

27 xx oo 30

können Sie Feld 30 wählen. Danach wandelt sich das Spielfeld wie folgt:

27 xx xx xx

Sinn des Spiels ist es, am Ende, d. h. wenn alle Felder mit Steinen belegt sind, über mehr Steine als der Gegner zu verfügen. Den Spielstand können Sie sich jeweils dann, wenn Sie am Zug sind, durch Eingabe von S vom Rechner ausgeben lassen.

Für jeden Spieler besteht Zugzwang, solange die Möglichkeit besteht, nach diesen Spielregeln Steine zu setzen. Ist dies nicht der Fall, müssen Sie durch Eingabe von P passen. Der Rechner überprüft dies genau. Stellt er fest, daß Sie noch an irgendeiner Stelle einen Stein setzen können, gibt er Ihnen die Meldung aus:

```
Du darfst nicht passen. Du kannst z. B. noch auf Feld xx
```

Sie können dann Ihre Eingabe wiederholen. Noch ein Tip: Die Felder am Rand und in den Ecken nehmen eine Schlüsselstellung ein. Versuchen Sie, dort so viele Steine wie möglich zu plazieren. Machen Sie keine Fehler, denn darauf spekuliert der Rechner . . .

Zum Programmaufbau:
Grundgerüst des Programms ist ein Feld, das in Programmzeile 70 definiert wird. Der Feldname ist A.

```
DIM A(64)
```

Mit diesen 64 Feldelementen kann jede einzelne Position auf dem Spielfeld erfaßt werden. Eine solche Position kann

1. leer sein (A(x)=0);
2. mit einem runden Stein besetzt sein (A(x)=1);
3. mit einem eckigen Stein besetzt sein (A(x)=−1).

Nach der Felddimensionierung wird der Bildschirm gelöscht und in Zeile 90 zur Programmzeile 530 verzweigt. Dadurch werden diverse Unterprogramme, die in den dazwischenliegenden Zeilen liegen, übersprungen.

Von Programmzeile 530 bis 900 werden Eingaben versorgt (Anzahl der Spieler und deren Namen etc.) und Informationstexte ausgegeben. Danach wird ab Programmzeile 920 das Spielfeld aufgebaut, und die Variablen werden mit Anfangswerten versehen (bis Programmzeile 1080). Je nachdem, ob der Wert der Variablen F 1 oder −1 ist, ist entweder der erste Spieler oder der zweite (der auch der Rechner sein kann) am Zug. In Programmzeile 1550 kommt nach jedem abgeschlossenen Zug durch

F=−F

wieder der Gegenspieler an die Reihe. An dieser Stelle enden unsere Erläuterungen. Dies hat folgenden Grund: Der übrige Teil der BASIC-Routinen behandelt im wesentlichen die Strategie des Rechners und diverse Ein-/Ausgabefunktionen. Ein Spiel wird einfach uninteressant, wenn man die Strategie seines Gegners kennt, weil man die BASIC-Befehle nachvollziehen kann. Viel interessanter hingegen ist es, anhand des Spielverhaltens eines Rechners dessen Strategie zu erkennen und dann zu gewinnen.

Viel Spaß mit REVERSI!

```
10      'ANGEPASST AUF COLOUR GENIE
30      CLS
40      PRINT@480," R E V E R S I"
50      FORN=1TO650:NEXT
70      RANDOM:DIMA(64):CLS:GOTO220
80      FORX=1TO1200:NEXTX:RETURN
90      IFF=1GOTO110
100     PRINT@V*40+H,;:PRINT"THENTHEN";:RETURN
110     PRINT@V*40+H,;:PRINT"MKD$VAL";:RETURN
120     GOTO650
130     H=4*(X-8*INT(X/8.1)):V=6+2*INT(X/8.1):RETURN

140     FOR TT=0 TO 200 STEP 40:PRINT@TT,CHR$(30);:N
        EXT:PRINT@0,;:RETURN
150     D=243:N=1:GOSUB170:GOSUB200:GOSUB200:GOSUB20
        0:GOSUB200:GOSUB200:GOSUB200
160     GOSUB200:RETURN
170     FORZ=1TO8:PRINT@INT(D),N:D=D+4:N=N+1:NEXTZ:R
        ETURN
180     RETURN
200     D=D+48:GOSUB170:RETURN
210     D=D+76:GOSUB170:RETURN
```

Abb. 7.2: Die Programmliste zu Reversi

```
220   PRINTCHR$(23);:K=0:K=RND(2):PRINT@20,"
       ":B$="Der Rechner "
230   PRINT:INPUT"1 oder 2 Spieler";J:IFJ<>1ANDJ<>
      2THEN230ELSEPRINT:INPUT"Deinen Namen bitte :
      ";A$
240   F=1:IFJ=2GOTO270
250   CLS:IFK=1GOTO290
260   GOTO280
270   PRINT:INPUT"und den Namen des Gegners bitte
      :";B$:CLS
280   PRINT@ 40,B$;:GOTO300
290   PRINT@ 40,A$;:GOTO300
300   PRINT" soll anfangen und spielt
      mit folgenden Steinen :"
310   H=30:V=2:F=1:GOSUB90
320   IFK=1GOTO340
330   PRINT@160,A$;:GOTO350
340   PRINT@160,B$;
350   PRINT" spielt mit
      folgenden Steinen"
360   V=5:F=-1:GOSUB90
370   GOSUB80:CLS
380   PRINT"Durch Eingabe von 'S' kannst Du
      dir den Spielstand anzeigen lassen":GOSUB80
390   CLS:GOTO440
400   REM
410   FORX=1TO64:IFA(X)=0GOTO430
420   GOSUB130:F=A(X):GOSUB90
430   NEXTX:RETURN
440   FORX=1TO64:A(X)=0:NEXTX
450   A(28)=1:A(37)=1:A(29)=-1:A(36)=-1
460   GOSUB150:GOSUB180:GOSUB400:GOTO860
470   GOSUB140:PRINT@1,A$;", ";:GOTO520
480   GOSUB140:PRINT@1,B$;", ";:GOTO520
490   GOSUB 140:PRINT"Ich bin dran,ich glaub' ich
      nehm "
500   GOSUB980:RESTORE:IFR=0GOTO470
510   GOSUB140:PRINT@1," ";:PRINT"Okay, gemacht' I
      ch nehme Feld ";T;:Z=T:E=0:GOTO590
520   I$="":Z$="":PRINT@80,"";:INPUT"Waehle ein Fe
      ld oder gib <P> ein, wenn
      Du bei diesem Zug passt";Z$:GOSUB 140:IFZ$="
      S"THENGOSUB1331:GOSUB80
530   Z=VAL(Z$):IFZ$<>"S"GOTO550
540   PRINT@0,"":GOSUB140:GOSUB180:GOTO520
550   GOSUB140:IFZ$="P"GOTO810
560   IF(Z>0)*(Z<65)GOTO580
570   PRINT@1,"So ein Feld gibt es nicht.":GOSUB80
      :GOSUB140:GOTO520
580   IFA(Z)<>0GOTO610
590   W=0:L=0:E=0:GOSUB620:IFW=0GOTO520
600   F=-F:GOTO860
610   PRINT@1,"Feld schon belegt.":GOSUB80:GOSUB14
      0:GOTO520
620    D=1:GOSUB120:D=-1:GOSUB120:FORD=7TO9:GOSUB1
      20:NEXTD:FORD=-9TO-7:GOSUB120:NEXTD
630   IFE<>1GOTO1270
640   RETURN
```

Abb. 7.2: Die Programmliste zu Reversi (Fortsetzung)

```
650   G=Z+D
660   IFG<1RETURN
670   IFG>64RETURN
680   IFA(G)=FTHENRETURN
690   IFA(G)=0RETURN
700   IFABS(D)=8GOTO730
710   IFINT(G/8)=(G/8)RETURN
720   IFINT((G-1)/8)=((G-1)/8)RETURN
730   G=G+D:IFG<1RETURN
740   IFG>64RETURN
750   IFA(G)=FGOTO770
760   GOTO690
770   Y=0:W=1:IFE=1GOTO800
780   IFL=1RETURN
790   FORX=G-DTOZ+DSTEP-D:A(X)=F:GOSUB130:GOSUB90:
      NEXTX:RETURN
800   FORX=G-DTOZ+DSTEP-D:Q=Q+1:NEXTX:RETURN
810   W=0:L=1:FORZ=1TO64:IFA(Z)<>0GOTO830
820   GOSUB620:IFW=1GOTO850
830   NEXTZ:IFY=1GOTO1330
840   Y=1:F=-F:GOTO860
850   PRINT@1,"Du darfst nicht passen.
      Du kannst z.B. noch auf Feld ";Z:GOSUB80:GOT
      O520
860   IFJ=1GOTO900
870   IFF=1GOTO930
880   IFK=1GOTO480
890   GOTO470
900   IFF=1GOTO950
910   IFK=1GOTO490
920   GOTO470
930   IFK=1GOTO470
940   GOTO480
950   IFK=1GOTO470
960   GOTO490
970   CLS:GOTO460
980   R=0:T=0:E=1
990    FORP=28TO1STEP-1:IFA(P)<>0I=P-9
1000   NEXTP:FORP=37TO64:IFA(P)<>0O=P+9
1010   NEXTP:FORP=1TO4:READZ:IFA(Z)<>0GOTO1040
1020  IF(Z<I)+(Z>O)GOTO1040
1030  GOSUB1210:IFQ<>0GOTO1180
1040  NEXTP:R=0:FORP=1TO12:READA,B,N,M:FORZ=ATOBST
      EPN
1050  IFA(Z)<>0GOTO1130
1060   IF(Z<I)+(Z>O)GOTO1140
1070   GOSUB1210:IFQ=0GOTO1130
1080  IFQ*M>RGOTO1110
1090  IFQ*M=RGOTO1120
1100  GOTO1130
1110  R=Q*M:T=Z:GOTO1130
1120  IFRND(3)=2T=Z
1130  NEXTZ
1140  NEXTP:IFR<>0RETURN
1150   FORP=1TO12:READZ:IFA(Z)<>0GOTO1170
1160   GOSUB1210:IFQ<>0GOTO1180
1170  NEXTP:GOTO1190
1180   R=Q:T=Z:RETURN
```

Abb. 7.2: Die Programmliste zu Reversi (Fortsetzung)

```
1190  PRINT@1,"Ich muss passen.":IFY=1GOTO1330
1200  Y=1:F=-F:GOSUB80:RETURN
1210  N=T:IFT=0N=Z
1220  Q=0:PRINT@34,N:GOSUB620:RETURN
1230  DATA1,8,57,64,3,6,1,3,24,48,8,3,59,62,1,3,1
      7,41,8,3
1240  DATA11,14,1,1,23,47,8,1,51,54,1,1,18,42,8,1

1250  DATA19,22,1,2,27,30,1,2,35,38,1,2,43,46,1,2

1260  DATA2,16,63,49,7,56,58,9,10,15,55,50
1270  IFL=1RETURN
1280  IFW=1GOTO1300
1290  PRINT@1,"Ein solcher Zug waere gegen
       die Spielregeln.":GOSUB80:GOSUB140:RETURN
1300  GOSUB130:GOSUB90:A(Z)=F
1310  FORX=1TO64:IFA(X)=0GOTO1380
1320  NEXTX
1330  GOSUB1331:GOTO1370
1331  C=0:D=0:FORX=1TO64:IFA(X)=1THENC=C+1
1340  IFA(X)=-1THEND=D+1
1350  NEXTX
1360  GOSUB140:PRINT@0," ■   HAT ";D;" STEINE
      ":PRINT" ●   HAT ";C;" STEINE":RETURN
1370  Y$=INKEY$:IFY$=""THEN1370ELSECLS:END
1380  GOSUB80:RETURN
```

Abb. 7.2: Die Programmliste zu Reversi (Fortsetzung)

7.2 BÖRSENSPIEL

Beim Börsenspiel können bis zu drei Personen mitspielen. Ziel des Spiels ist es, durch geschickten An- und Verkauf einer begrenzten Anzahl verfügbarer Aktien so schnell wie möglich zum Millionär zu werden. Hierbei steht es jedem Spieler frei, sich von der Bank Kredite zum Ankauf von Aktien geben zu lassen oder durch Verkäufe bei Hochnotierungen die notwendigen Gelder zur Erlangung des einmaligen Reichtums aufzubringen.

Sobald einmal jeder Spieler seine Transaktionen vorgenommen hat, ändern sich die Tageskurse der Aktien nach Angebot und Nachfrage.

Nach jeder Runde wird dem Spieler sein Vermögens- und Bargeldbestand angezeigt und eine Bilanz der durchgeführten Transaktionen erstellt.

```
British Petrol 100 DM 297 Stk
Volkswagen     100 DM 297 Stk
Chemie Hoechst 100 DM 297 Stk
Bayer AG       100 DM 297 Stk
=========================================

Aktien von British Petrol :     1 Stk
Aktien von Volkswagen     :     1 Stk
Aktien von Chemie Hoechst :     1 Stk
Aktien von Bayer AG       :     1 Stk
Gesamtwert der Aktien :       400 DM
Bargeld               :       300 DM
Schulden an die Bank  :         0 DM
-----------------------------------------

Gesamtvermoegen       :       700 DM

KALLE, nehmen Sie Transaktionen vor?
```

Abb. 7.3: So sieht das Spielfeld im Börsenspiel aus

```
9     ' BOERSENSPIEL, ANGEPASST AUF COLOUR-GENIE
10    CLEAR 1000
20    DATA"British Petrol ","Volkswagen     ","Che
      mie Hoechst ","Bayer AG       "
30    S$="### Stk"
40    K$="####### DM"
50    P=680
60    P$=CHR$(30)
70    M$=" -## DM"
80    B$=" +## DM"
90    DM$="### DM "
100   NN$="  %          %"
110   CLS
120   PRINT @ 121,"B O E R S E N S P I E L"
130   PRINT @ 161,"======================="
140   PRINT @ 240,"Wieviel Spieler (1-3)";
150   INPUT X
160   IF X<1 OR X>3 THEN 140
170   X=INT(X)
180   CLS
190   FOR I=1 TO X
200   PRINT"Name von Spieler Nr." I;
210   INPUT N$(I)
220   N$(I)=LEFT$(N$(I), 11)
230   K(I)=300
```

Abb. 7.4: Die Programmliste zum Börsenspiel

```
240   NEXT I
250   FOR I=1 TO 4
260   READ G$(I)
270   W(I)=100
280   A(I)=300-X
290   FOR J=1 TO X
300   S(J, I)=1
310   NEXT J, I
320   Z%=Z%+1
330   FOR I=1 TO X
340   CLS
350   GOSUB 370
360   GOTO 600
370   FOR J=1 TO 4
380   PRINT G$(J); USING DM$; W(J);
390   PRINT USING S$; A(J);:PRINT"    ";
400   IF M%(J)=1 THEN PRINT USING M$; D%(J); ELSE
      IF M%(J)=-1 PRINT USING B$; D%(J);
410   PRINT
420   NEXT
430   PRINT STRING$(40,"=")
440   FOR J=1 TO 4
450   PRINT"Aktien von ";G$(J);":",USING S$; S(I,
      J);
460   PRINT Q$(I)
470   NEXT J
480   T(I)=0
490   FOR K=1 TO 4
500   T(I)=T(I)+S(I, K)*W(K)
510   NEXT
520   PRINT"Gesamtwert der Aktien :     ";USING K$;
       T(I)
530   PRINT"Bargeld               :     ";USING K$;
       K(I)
540   V(I)=V(I)+(V(I)*.01)
550   H(I)=K(I)+T(I)-V(I)
560   PRINT"Schulden an die Bank  :     ";USING K$;
       V(I)
570   PRINT STRING$(40,"-")
580   PRINT"Gesamtvermoegen       :     ";USING K$
      ; H(I)
590   RETURN
600   PRINT @ P, N$(I);", nehmen Sie Transaktionen
       vor?";P$
610   Z$=INKEY$
620   IF Z$="" THEN 610
630   IF Z$="N" OR Z$="n" THEN 1090
640   IF Z$<>"J" AND Z$<>"j" THEN 610
650   PRINT @ P, P$;"Wollen Sie Aktien kaufen?";
660   Z$=INKEY$
670   IF Z$="" THEN 660
680   IF Z$="n" OR Z$="N" THEN 880
690   IF Z$<>"J" AND Z$<>"j" THEN 660
700   PRINT @ P, P$;
710   G=0
720   INPUT"Welche Gesellschaft (1-4)";G
730   IF G<0 OR G>4 THEN 700
740   G=INT(G)
```

Abb. 7.4: Die Programmliste zum Börsenspiel (Fortsetzung)

```
750   IF G=0 THEN 880
760   PRINT @ P, P$;"Wie viele Aktien von " G$(G)"

      wollen Sie kaufen";
770   INPUT A
780   IF A<0 THEN 760
790   A=INT(A)
800   IF A>A(G)PRINT @ P, P$;"So viele Aktien sind
       nicht
      erhaeltlich!";: FOR U=1 TO 500: NEXT U: GOTO
       760
810   IF W(G)*A>K(I)PRINT @ P, P$;"Sie haben nicht
       genuegend Kapital!": FOR U=1 TO 500: NEXT U
      : GOTO 760
820   S(I, G)=S(I, G)+A
830   A(G)=A(G)-A
840   K(I)=K(I)-W(G)*A
850   PRINT CHR$(28);
860   GOSUB 370
870   GOTO 700
880   PRINT @ P, P$;"Wollen Sie Aktien verkaufen?"
      ;
890   Z$=INKEY$
900   IF Z$="" THEN 890
910   IF Z$="N" OR Z$="n" THEN 1090
920   IF Z$<>"J" AND Z$<>"j" THEN 890
930   PRINT @ P, P$;
940   G=0
950   INPUT"Welche Gesellschaft (1-4)";G
960   IF G<0 OR G>4 THEN 930
970   G=INT(G)
980   IF G=0 THEN 1090
990   PRINT @ P, P$;"Wie viele Aktien von " G$(G)"

      wollen Sie verkaufen";
1000  INPUT A
1010  IF A<0 THEN 990
1020  IF A>S(I, G)PRINT @ P, P$;"Soviele Aktien ha
      ben Sie nicht!": FOR U=1 TO 500: NEXT U: GOT
      O 990
1030  S(I, G)=S(I, G)-A
1040  A(G)=A(G)+A
1050  K(I)=K(I)+W(G)*A
1060  PRINT CHR$(28);
1070  GOSUB 370
1080  GOTO 930
1090  PRINT @ P, P$;"Wollen Sie zur Bank?";
1100  Z$=INKEY$
1110  IF Z$="" THEN 1100
1120  IF Z$="j" OR Z$="J" THEN 1400
1130  IF Z$<>"N" AND Z$<>"n" THEN 1100
1140  NEXT I
1150  GOSUB 1660
1160  FOR I=1 TO 4
1170  F=0
1180  M=10*RND(12)-60
1190  W(I)=W(I)+M
1200  IF W(I)>=10 THEN 1280
```

Abb. 7.4: Die Programmliste zum Börsenspiel (Fortsetzung)

```
1210 F=1
1220 M%(I)=1
1230 D%(I)=10-W(I)
1240 FOR J=1 TO X
1250 IF S(J, I)<>0 THEN Y=(10-W(I))*S(J, I): K(J)
     =K(J)-Y:IF K(J)<0 THEN K(J)=K(J)+Y: V(J)=V(J
     )+Y
1260 NEXT J
1270 W(I)=10
1280 IF W(I)<=250 THEN 1360
1290 F=1
1300 M%(I)=-1
1310 D%(I)=W(I)-250
1320 FOR J=1 TO X
1330 IF S(J, I)<>0 THEN Y=(W(I)-250)*S(J, I): V(J
     )=V(J)-Y:IF V(J)<0 THEN K(J)=K(J)+ABS(V(J)):
      V(J)=0
1340 NEXT J
1350 W(I)=250
1360 IF F=0 THEN M%(I)=0
1370 NEXT I
1380 CLS
1390 GOTO 320
1400 IF V(I)=0 THEN 1540
1410 PRINT @ P, P$;"Wollen Sie Kredit zurueckzahl
     en?";
1420 Z$=INKEY$
1430 IF Z$="" THEN 1420
1440 IF Z$="N" OR Z$="n" THEN 1540
1450 IF Z$<>"J" AND Z$<>"j" THEN 1420
1460 PRINT @ P, P$;
1470 INPUT"Wieviel Geld wollen Sie zurueckzahlen"
     ;Z
1480 IF Z>V(I)OR Z<0 THEN 1460
1490 IF Z>K(I)PRINT @ P, P$;"Sie haben nicht genu
     egend Kapital": FOR U=1 TO 500: NEXT U: GOTO
      1460
1500 V(I)=V(I)-Z
1510 K(I)=K(I)-Z
1520 PRINT CHR$(28);
1530 GOSUB 370
1540 PRINT @ P, P$;"Wollen Sie Kredit nehmen?";
1550 Z$=INKEY$
1560 IF Z$="" THEN 1550
1570 IF Z$="N" OR Z$="n" THEN 1140
1580 IF Z$<>"J" AND Z$<>"j" THEN 1550
1590 PRINT @ P, P$;
1600 INPUT"Wieviel Kredit wollen Sie nehmen";Z
1610 IF Z<0 THEN 1590
1620 IF V(I)+Z>25000 PRINT @ P, P$;"Soviel Kredit
      koennen Sie nicht nehmen'": FOR U=1 TO 500:
      NEXT: GOTO 1140
1630 V(I)=V(I)+Z
1640 K(I)=K(I)+Z
1650 GOTO 1140
1660 CLS
1670 PRINT"Bilanz nach" Z%"Runden:"
1680 PRINT
```

Abb. 7.4: Die Programmliste zum Börsenspiel (Fortsetzung)

```
1690  PRINT"Spieler    ";
1700  FOR I=1 TO X
1710  PRINTUSING NN$;N$(I);
1720  NEXT I
1730  PRINT
1740  PRINT STRING$(X*10+10,"=")
1750  PRINT"Aktienwert";
1760  FOR I=1 TO X
1770  PRINT USING K$; T(I);
1780  NEXT
1790  PRINT
1800  PRINT"Bargeld    ";
1810  FOR I=1 TO X
1820  PRINT USING K$; K(I);
1830  NEXT
1840  PRINT
1850  PRINT"Schulden   ";
1860  FOR I=1 TO X
1870  PRINT USING K$; V(I);
1880  NEXT
1890  PRINT
1900  PRINT STRING$(X*10+10,"-")
1910  PRINT"Vermoegen ";
1920  FOR I=1 TO X
1930  PRINT USING K$; H(I);
1940  NEXT
1950  PRINT
1960  MX=0
1970  FOR I=1 TO X
1980  IF H(I)>MX THEN MX=H(I): MY=I
1990  NEXT
2000  IF MX<100000 THEN PRINT: PRINT"Druecke ENTER
      ";: INPUT Z$: RETURN
2010  PRINT
2020  PRINT"Der Sieger ist:   ";N$(MY)
2030  FOR I=1 TO 2000
2040  NEXT
2050  END
```

Abb. 7.4: Die Programmliste zum Börsenspiel (Fortsetzung)

Anhang A

Treiberprogramm für die DEFFN-Funktion unter Kassettenbetrieb

In Kapitel 3 wurde die Anwendung von Funktionen, die mit Hilfe der DEFFN-Anweisung definiert werden, behandelt.

Der sich im Rechner befindende BASIC-Interpreter des Colour-Genies unterstützt die DEFFN-Anweisung genauso wie die übrigen unter Kap. 6 aufgeführten Anweisungen erst, wenn durch Einstecken des Disk-BASIC-Einschubs die entsprechenden Interpreter-Module zur Verfügung gestellt werden.

Man kann solche Treiber-Module jedoch auch im RAM-Speicher eines Rechners ablegen. Man muß dann nur sicherstellen, daß dieser RAM-Speicher „geschützt“ wird, d. h. nicht mit irgendwelchen Daten überschrieben werden darf.

Das im folgenden vorgestellte Programm lädt sich in den Speicherbereich der frei programmierbaren Zeichen und startet sich danach selbst. Demzufolge dürfen Sie, solange Sie mit DEFFN arbeiten, keine frei programmierbaren Zeichen definieren.

Durch Betätigung der beiden RESET-Tasten können Sie jederzeit das Programmodul wieder desaktivieren.

```
10   CLS
20   PRINT"Mit Hilfe dieses Programms sind
     Colour-Genie Besitzer ohne Disketten-
     station in der Lage, unter Microsoft-
     BASIC mit der Funktion DEFFN zu ar-
     beiten."
30   PRINT"Ein dazu notwendiges Treiber-
     programm in Maschinensprache wird da-
     zu in den Speicher der frei pro-
     grammierbaren Zeichen geladen, mit
     denen dann nicht mehr gearbeitet wer-
     den darf."
```

Abb. A.1: Das Treiberprogramm für DEFFN unter Kassettenbetrieb

```
40      A=&HF400:'Anfang des Treibers
50      E=&HF528:'Endadresse des Treibers
60      FOR X=A TO E
70      READ W:POKE X,W
80      SU=SU+W
90      NEXT X
100     IF SU <> 37624 THEN PRINT"
        EINGABEFEHLER IN DEN DATA-ZEILEN":STOP
110     PRINT:PRINT"Treiberprogramm eingebunden."
120     FOR X=1TO100:NEXTX
130     CALL F400
140     DATA  33, 127
150     DATA  244, 34, 86, 65, 33
160     DATA  47, 244, 34, 92, 65
170     DATA  195, 193, 29, 183, 194
180     DATA  214, 9, 201, 215, 62
190     DATA  128, 50, 220, 64, 182
200     DATA  71, 205, 18, 38, 24
210     DATA  8, 62, 128, 50, 220
220     DATA  64, 205, 13, 38, 58
230     DATA  175, 64, 254, 3, 201
240     DATA  254, 190, 32, 25, 205
250     DATA  20, 244, 205, 40, 40
260     DATA  125, 18, 19, 124, 18
270     DATA  62, 40, 190, 194, 5
280     DATA  31, 215, 205, 13, 38
290     DATA  62, 44, 24, 244, 254
300     DATA  193, 194, 151, 25, 205
310     DATA  96, 244, 213, 207, 213
320     DATA  205, 2, 43, 227, 115
330     DATA  35, 114, 225, 201, 17
340     DATA  236, 67, 215, 208, 214
350     DATA  48, 135, 131, 95, 215
360     DATA  201, 70, 33, 94, 244
370     DATA  229, 197, 205, 41, 244
380     DATA  245, 204, 218, 41, 235
390     DATA  33, 33, 65, 241, 201
400     DATA  205, 20, 244, 245, 175
410     DATA  245, 235, 126, 35, 182
420     DATA  111, 124, 181, 202, 74
430     DATA  30, 26, 254, 40, 32
440     DATA  63, 207, 40, 213, 205
450     DATA  33, 244, 193, 34, 235
460     DATA  67, 32, 9, 213, 245
470     DATA  197, 213, 205, 136, 40
480     DATA  24, 16, 237, 68, 111
490     DATA  38, 255, 57, 249, 237
500     DATA  68, 213, 245, 197, 213
510     DATA  205, 15, 244, 209, 42
520     DATA  235, 67, 227, 215, 205
530     DATA  43, 31, 209, 26, 190
540     DATA  194, 151, 25, 235, 254
550     DATA  41, 40, 4, 207, 44
560     DATA  24, 197, 19, 215, 207
570     DATA  213, 213, 205, 55, 35
580     DATA  43, 215, 194, 151, 25
590     DATA  225, 34, 235, 67, 231
600     DATA  32, 50, 237, 91, 179
```

Abb. A.1: Das Treiberprogramm für DEFFN unter Kassettenbetrieb (Fortsetzung)

```
610    DATA  64, 42, 33, 65, 27
620    DATA  27, 27, 223, 40, 5
630    DATA  205, 67, 40, 24, 32
640    DATA  237, 91, 211, 64, 205
650    DATA  245, 41, 34, 179, 64
660    DATA  1, 3, 0, 237, 176
670    DATA  24, 15, 209, 254, 3
680    DATA  40, 238, 33, 0, 0
690    DATA  57, 235, 205, 15, 244
700    DATA  235, 249, 241, 183, 32
710    DATA  237, 231, 204, 194, 41
720    DATA  42, 235, 67, 43, 215
730    DATA  241, 195, 25, 40,0
```

Abb. A.1: Das Treiberprogramm für DEFFN unter Kassettenbetrieb (Fortsetzung)

Anhang B

Wichtige Speicheradressen

In diesem Abschnitt sollte ursprünglich ein anderes Thema behandelt werden.Daß hier schließlich doch die Adressen von Interpreterroutinen und vom Kommunikationsbereich behandelt werden, hat mehrere Gründe. Erstens kann es vorkommen, daß Ihnen irgendwann Software in die Hände fällt, in der über PEEK und POKE bestimmte Adressen dieser Art angesprochen werden, so daß ein Nachschlagewerk für das Verständnis eines solchen Programms wünschenswert ist. Zweitens werden auch in diesem Buch solche Adressen (wie im Beispiel „Ermittlung der gegenwärtigen Cursor-Position auf dem Bildschirm") verwendet, zu denen man im BASIC-Handbuch keinerlei Dokumentation findet.

Das liegt in erster Linie daran, daß die Daten, die sich in solchen Speicheradressen befinden, nicht für die Nutzung durch Anwender, sondern als Informationsbausteine für den Interpreter vorgesehen sind. So muß der Interpreter z. B. wissen, mit welcher Adresse ein sich im Speicher befindendes BASIC-Programm beginnt, um dieses abarbeiten zu können.

Für Sie kann diese Information dann von Nutzen sein, wenn Sie solche rechnerinternen Vorgänge nachvollziehen wollen.

Im folgenden finden Sie eine Tabelle solcher Speicheradressen mit einigen Anwendungsbeispielen:

Adresse HEX	Adresse DEZ	Bedeutung
3800–380F	14336–14351	CRTC-Controller: Werte für den LGR-Bildschirm (detaillierte Beschreibung siehe Handbuch Seite 115ff.)
3810–381F	14352–14367	CRTC-Controller: Werte für den FGR-Bildschirm bei PAL-Monitor
3823–3832	14371–14386	CRTC-Controller: Werte für den LGR-Bildschirm (NTSC-Norm)
3833–3842	14387–14402	CRTC-Controller: Werte für den FGR-Bildschirm (NTSC-Norm)
4020–4021	16417–16417	Cursoradresse

Adresse HEX	Adresse DEZ	Bedeutung
4022	16418	Cursorzeichen (ASCII-Code)
4023	16419	Letzter Farbencode im LGR-Modus
4024	16420	Letztes gedrücktes Zeichen für REPEAT
4028	16424	Drucker: Anzahl der Zeilen pro Seite
4029	16425	Drucker: Zeilenzähler
4036	16438	Tastaturstatus für Tastaturzeile F801
4037	16439	Tastaturstatus für Tastaturzeile F802
4038	16440	Tastaturstatus für Tastaturzeile F804
4039	16441	Tastaturstatus für Tastaturzeile F808
403A	16442	Tastaturstatus für Tastaturzeile F810
403B	16443	Tastaturstatus für Tastaturzeile F820
403C	16444	Tastaturstatus für Tastaturzeile F840
403D	16445	Tastaturstatus für Tastaturzeile F880 (s. S. 125 BASIC-Handbuch)
4099	16537	INKEY-Zwischenspeicher
409A	16538	Letzter Fehlercode für ERR
409B	16539	Druckkopfposition
409C	15540	Ausgabekennung: 0=Video, 1=Drucker, 2=Kass.
409D	16541	Zeilenlänge auf dem Bildschirm
40A2	16546	aktuelle Zeilennummer
40A4–40A5	16548–16549	Speicheradresse, von der ab ein Programm im Speicher abgelegt wurde (Berechnung: ADR=PEEK(&H40A4)+256XPEEK(&H40A5))
40A6	16550	Position des Cursors in der Bildschirmzeile
40B1–40B2	16561–16562	Letzter im BASIC verfügbarer Speicherplatz
40DA–40DB	16602–16603	Aktuelle DATA-Zeile für READ-Anweisung
40EA	16618	Zeile, in der der letzte Fehler auftrat
40EC	16620	Zeilennummer für EDIT.
40EE	16622	Zeigt auf die Anweisung, bei deren Ausführung der letzte Fehler auftrat.
42F0–430F	17138–17167	CRTC-Tabelle: wird von 3800 hierhin kopiert.
4310–4312	16168–17170	Aufzeichnungsrate für Programme und Daten auf Kassette
4313	17171	Code der FCOLOUR-Farbe, mit der gerade gezeichnet wird
4314	17172	Letzter verwendeter SCALE-Faktor
4390–439F	17296–17311	Farbencodetabelle

Wie man einige der Funktionen sinnvoll in eigenen Programmen anwenden kann, entnehmen Sie den Abbildungen B.1 und B.2.

```
10    CLS:CLEAR1000
20    PRINT"AUS DEN SPEICHERADRESSEN KANN FOL-
      GENDES ENTNOMMEN WERDEN:":PRINT:PRINT
30    P=PEEK(&H4023)+1
40    PRINT"DIE ZULETZT VON IHNEN FUER DEN LGR-
      BILDSCHIRM ANGEWAEHLTE FARBE HATTE DEN
      FARBENCODE ";P;", ALSO COLOUR ";P
50    P=PEEK(&H4024)
60    PRINT"DAS ZULETZT VON IHNEN AUF DER TAS-
      TATUR BETAETIGTE ZEICHEN HAT DEN ASCII-CODE
      :";P
70    P=PEEK(&H409A)
80    IF P=0 THEN A$="KEIN FEHLER" ELSE A$="EIN FE
      HLER MIT DEM FEHLERCODE ERR="+STR$(P)
90    PRINT"IM PROGRAMM TAUCHTE ";A$;" AUF."
100   P=PEEK(&H40A4)+256*PEEK(&H40A5)
110   PRINT"DIESES PROGRAMM WURDE VOM RECHNER AB S
      PEICHERADRESSE ";P;" ABGELEGT."
120   P=PEEK(&H40A6)
130   PRINT"DER CURSOR BEFINDET SICH Z.ZT. IN BILD
      SCHIRMSPALTE ";P
140   P=PEEK(&H40B1)+256*PEEK(&H40B2)
150   PRINT"DER LETZTE Z.ZT. VERWENDBARE SPEICHERP
      LATZ HAT DIE ADRESSE ";P
```

Abb. B.1: Bestimmte Speicheradressen können wertvolle Informationen enthalten, die Sie nicht direkt über eine BASIC-Anweisung abrufen können

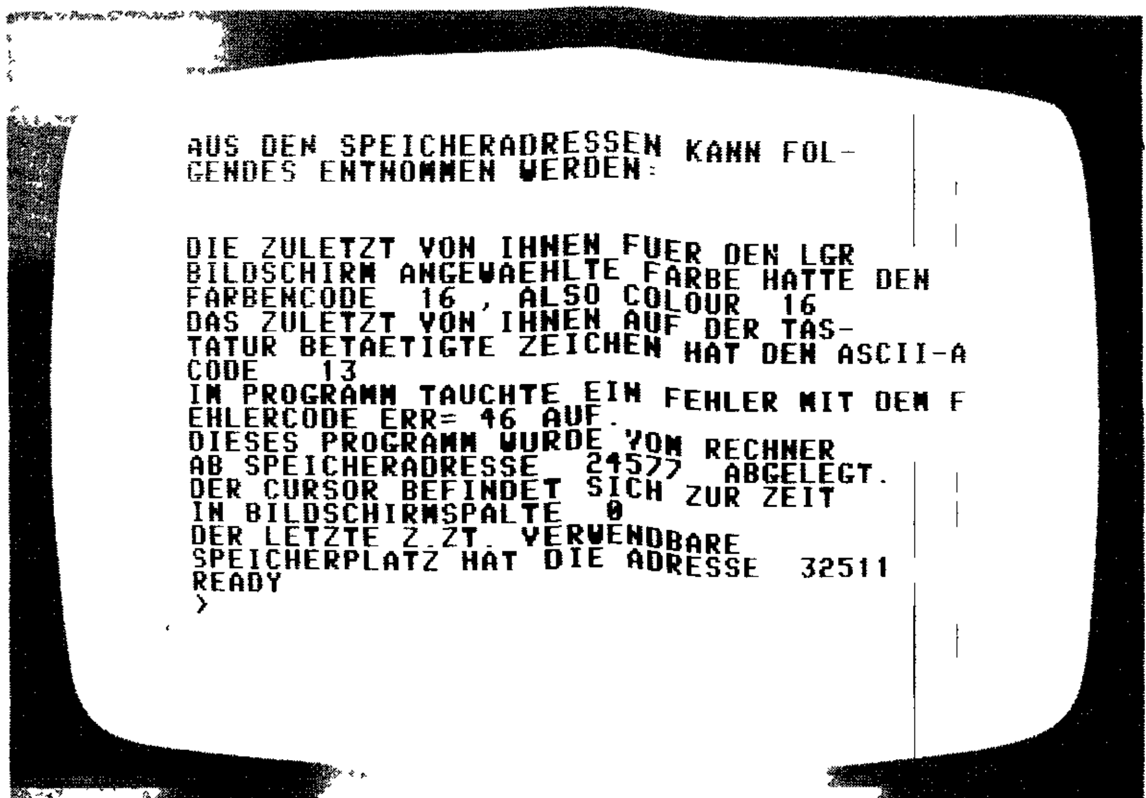


Abb. B.2: Bildschirmausgabe zum Programm in Abb. B.1

Anhang C

Im folgenden finden Sie einige Zeichensätze, die Sie mit Hilfe des im Kapitel 2 vorgestellten Hilfsprogramms in den Speicher für frei definierbare Zeichen laden können. An das Ende der DATA-Zeilen wird das Programm aus Abb. 2.17 angehängt und durch RUN gestartet. Die im folgenden vorgestellten Schrifttypen tragen die Namen

- COUNT
- BLIPPO-BLACK
- GOTHIC
- COLOSSAL
- BYTE
- STOP
- PUDGY
- PINOCCHIO
- OUTLINE

```
32 REM SCHRIFTTYPE  COUNT
33 DATA 0,0,246,246,0,0,0,0               :REM !
34 DATA 0,160,192,0,160,192,0,0           :REM "
35 DATA 100,254,254,100,254,100,0,0       :REM #
36 DATA 40,100,246,222,76,40,0,0          :REM $
37 DATA 98,100,8,16,38,70,0,0             :REM %
38 DATA 92,254,254,162,138,76,0,0         :REM &
39 DATA 0,0,160,192,0,0,0,0               :REM '
40 DATA 0,56,124,238,198,0,0,0            :REM (
41 DATA 0,198,238,124,56,0,0,0            :REM )
42 DATA 84,56,254,254,56,84,0,0           :REM *
43 DATA 0,24,60,60,24,0,0,0               :REM +
44 DATA 0,0,5,6,0,0,0,0                   :REM ,
45 DATA 0,24,24,24,24,24,0,0              :REM -
46 DATA 0,6,6,0,0,0,0,0                   :REM .
47 DATA 2,4,8,16,32,64,128,0              :REM /
48 DATA 126,254,254,162,194,252,0,0       :REM 0
49 DATA 0,64,254,254,254,0,0,0            :REM 1
50 DATA 70,142,254,250,98,2,0,0           :REM 2
51 DATA 68,130,146,254,254,108,0,0        :REM 3
52 DATA 56,72,254,254,254,8,0,0           :REM 4
53 DATA 244,210,222,222,222,204,0,0       :REM 5
54 DATA 124,254,254,146,146,76,0,0        :REM 6
55 DATA 134,158,254,248,224,128,0,0       :REM 7
56 DATA 108,254,254,146,146,108,0,0       :REM 8
57 DATA 100,146,146,254,254,124,0,0       :REM 9
58 DATA 0,54,54,0,0,0,0,0                 :REM :
59 DATA 0,53,54,0,0,0,0,0                 :REM ;
60 DATA 0,16,56,124,238,198,0,0          :REM <
61 DATA 0,54,54,54,54,54,0,0              :REM =
62 DATA 195,231,126,60,16,0,0,0           :REM >
63 DATA 64,128,138,250,250,112,0,0        :REM ?
64 DATA 124,254,146,170,154,114,0,0       :REM @
65 DATA 126,200,200,254,254,126,0,0       :REM A
66 DATA 254,254,254,146,146,108,0,0       :REM B
67 DATA 124,254,254,130,130,68,0,0        :REM C
68 DATA 254,254,254,130,130,124,0,0       :REM D
69 DATA 124,254,254,146,146,130,0,0       :REM E
70 DATA 126,254,254,144,144,128,0,0       :REM F
71 DATA 124,254,254,130,138,76,0,0        :REM G
72 DATA 254,254,254,16,16,254,0,0         :REM H
73 DATA 0,0,254,254,254,0,0,0             :REM I
74 DATA 0,4,2,254,254,252,0,0             :REM J
75 DATA 254,254,254,16,16,238,0,0         :REM K
76 DATA 254,254,254,2,2,2,0,0             :REM L
77 DATA 126,254,192,126,192,126,0,0       :REM M
78 DATA 126,254,254,128,128,126,0,0       :REM N
79 DATA 124,254,254,130,130,124,0,0       :REM O
```

Abb. C.1: Der Schrifttyp COUNT

```
 80 DATA 254,254,254,144,144,96,0,0       :REM F
 81 DATA 124,254,254,130,130,127,1,0      :REM Q
 82 DATA 254,254,254,144,144,110,0,0      :REM R
 83 DATA 100,242,186,186,158,76,0,0       :REM S
 84 DATA 0,128,254,254,254,128,0,0        :REM T
 85 DATA 252,254,254,2,2,252,0,0          :REM U
 86 DATA 224,252,254,30,28,224,0,0        :REM V
 87 DATA 252,6,252,6,254,252,0,0          :REM W
 88 DATA 238,254,254,16,16,238,0,0        :REM X
 89 DATA 224,240,254,30,30,240,0,0        :REM Y
 90 DATA 134,142,158,242,226,194,0,0      :REM Z
 91 DATA 0,126,255,255,129,129,0,0        :REM [
 92 DATA 128,64,32,16,8,4,2,0             :REM \
 93 DATA 0,129,129,255,255,126,0,0        :REM ]
 94 DATA 8,24,48,16,8,0,0,0               :REM ^
 95 DATA 1,1,1,1,1,1,1,0                  :REM _
 96 DATA 0,0,224,208,0,0,0,0              :REM `
 97 DATA 28,34,34,60,62,30,0,0            :REM a
 98 DATA 252,254,254,34,34,28,0,0         :REM b
 99 DATA 28,62,62,34,34,20,0,0            :REM c
100 DATA 28,34,34,254,254,252,0,0         :REM d
101 DATA 28,62,62,42,42,18,0,0            :REM e
102 DATA 0,126,254,254,144,80,0,0         :REM f
103 DATA 25,37,37,63,63,30,0,0            :REM g
104 DATA 254,254,254,16,16,14,0,0         :REM h
105 DATA 0,0,190,190,190,0,0,0            :REM i
106 DATA 0,2,1,191,191,190,0,0            :REM j
107 DATA 254,254,254,16,16,110,0,0        :REM k
108 DATA 0,0,254,254,254,0,0,0            :REM l
109 DATA 30,62,48,30,48,30,0,0            :REM m
110 DATA 30,62,62,32,32,30,0,0            :REM n
111 DATA 28,62,62,34,34,28,0,0            :REM o
112 DATA 63,63,63,34,34,28,0,0            :REM p
113 DATA 28,34,34,63,63,63,0,0            :REM q
114 DATA 0,30,62,62,32,16,0,0             :REM r
115 DATA 20,50,42,42,38,20,0,0            :REM s
116 DATA 0,252,254,254,34,36,0,0          :REM t
117 DATA 60,62,62,2,2,60,0,0              :REM u
118 DATA 48,60,62,2,12,48,0,0             :REM v
119 DATA 60,6,60,6,62,60,0,0              :REM w
120 DATA 54,62,62,8,8,54,0,0              :REM x
121 DATA 57,5,5,63,63,62,0,0              :REM y
122 DATA 38,46,62,58,50,34,0,0            :REM z
123 DATA 0,16,126,255,195,195,0,0         :REM {
124 DATA 0,0,0,255,0,0,0,0                :REM |
125 DATA 0,195,195,255,126,16,0,0         :REM }
126 DATA 8,16,16,8,8,16,0,0               :REM ~
```

Abb. C.1: Der Schrifttyp COUNT (Fortsetzung)

```
32 REM SCHRIFTTYPE  BLIPPO BLACK
33 DATA 0,0,0,242,0,0,0,0               :REM !
34 DATA 0,0,224,0,0,224,0,0             :REM "
35 DATA 0,40,254,40,254,40,0,0          :REM #
36 DATA 0,36,84,254,84,72,0,0           :REM $
37 DATA 98,100,8,16,38,70,0,0           :REM %
38 DATA 0,12,114,138,100,10,0,0         :REM &
39 DATA 0,0,32,192,0,0,0,0              :REM '
40 DATA 0,0,56,68,130,0,0,0             :REM (
41 DATA 0,0,130,68,56,0,0,0             :REM )
42 DATA 0,84,56,254,56,84,0,0           :REM *
43 DATA 0,16,16,124,16,16,0,0           :REM +
44 DATA 0,0,1,6,0,0,0,0                 :REM ,
45 DATA 0,16,16,16,16,16,0,0            :REM -
46 DATA 0,0,6,6,0,0,0,0                 :REM .
47 DATA 2,4,8,16,32,64,0,0              :REM /
48 DATA 0,124,138,146,162,124,0,0       :REM 0
49 DATA 0,2,130,254,2,2,0,0             :REM 1
50 DATA 0,70,138,146,146,102,0,0        :REM 2
51 DATA 0,68,130,146,146,108,0,0        :REM 3
52 DATA 0,8,24,40,74,254,10,0           :REM 4
53 DATA 0,244,146,146,146,204,0,0       :REM 5
54 DATA 0,124,146,146,146,76,0,0        :REM 6
55 DATA 0,192,128,158,160,192,0,0       :REM 7
56 DATA 0,108,146,146,146,108,0,0       :REM 8
57 DATA 0,100,146,146,146,124,0,0       :REM 9
58 DATA 0,0,0,36,0,0,0,0                :REM :
59 DATA 0,0,1,38,0,0,0,0                :REM ;
60 DATA 0,16,40,68,130,130,0,0          :REM <
61 DATA 0,40,40,40,40,40,0,0            :REM =
62 DATA 0,130,130,68,40,16,0,0          :REM >
63 DATA 0,64,128,138,144,96,0,0         :REM ?
64 DATA 124,146,170,170,154,114,0,0     :REM @
65 DATA 2,62,82,144,82,62,2,0           :REM A
66 DATA 130,254,146,146,146,108,0,0     :REM B
67 DATA 0,124,130,130,130,68,0,0        :REM C
68 DATA 130,254,130,130,130,124,0,0     :REM D
69 DATA 130,254,146,186,130,198,0,0     :REM E
70 DATA 130,254,146,144,184,192,0,0     :REM F
71 DATA 0,124,130,130,146,92,16,0       :REM G
72 DATA 130,254,146,16,146,254,130,0 :REM H
73 DATA 0,198,130,254,130,198,0,0       :REM I
74 DATA 8,12,2,2,130,252,128,0          :REM J
75 DATA 130,254,18,40,68,130,130,0      :REM K
76 DATA 130,254,130,2,2,6,0,0           :REM L
77 DATA 130,254,66,48,66,254,130,0      :REM M
78 DATA 130,254,66,32,146,254,130,0     :REM N
79 DATA 0,124,130,130,130,124,0,0       :REM O
```

Abb. C.2: Der Schrifttyp BLIPPO BLACK

```
 80 DATA 130,254,146,144,144,96,0,0    :REM P
 81 DATA 0,124,130,138,132,122,0,0     :REM Q
 82 DATA 130,254,146,144,154,102,2,0   :REM R
 83 DATA 100,150,146,146,210,76,0,0    :REM S
 84 DATA 192,128,130,254,130,128,192,0:REM T
 85 DATA 128,252,130,2,130,252,128,0   :REM U
 86 DATA 128,248,132,2,132,248,128,0   :REM V
 87 DATA 128,254,4,24,4,254,128,0      :REM W
 88 DATA 130,198,40,16,40,198,130,0    :REM X
 89 DATA 128,224,146,14,146,224,128,0  :REM Y
 90 DATA 0,198,138,146,162,198,0,0     :REM Z
 91 DATA 0,254,130,130,130,0,0,0       :REM [
 92 DATA 64,32,16,8,4,2,0,0            :REM \
 93 DATA 0,130,130,130,254,0,0,0       :REM ]
 94 DATA 0,32,64,128,64,32,0,0         :REM ^
 95 DATA 1,1,1,1,1,1,1,0               :REM _
 96 DATA 0,0,128,64,32,0,0,0           :REM `
 97 DATA 4,42,42,42,28,2,0,0           :REM a
 98 DATA 130,254,20,34,34,28,0,0       :REM b
 99 DATA 0,28,34,34,34,20,0,0          :REM c
100 DATA 0,28,34,34,20,254,130,0       :REM d
101 DATA 0,28,42,42,42,26,0,0          :REM e
102 DATA 0,18,126,146,144,64,0,0       :REM f
103 DATA 0,25,37,37,25,62,0,0          :REM g
104 DATA 130,254,18,32,34,30,2,0       :REM h
105 DATA 0,0,34,190,2,0,0,0            :REM i
106 DATA 0,2,1,1,33,190,0,0            :REM j
107 DATA 130,254,4,8,52,34,0,0         :REM k
108 DATA 0,0,130,254,2,0,0,0           :REM l
109 DATA 34,62,32,62,32,30,0,0         :REM m
110 DATA 34,62,16,32,32,30,0,0         :REM n
111 DATA 0,28,34,34,34,28,0,0          :REM o
112 DATA 33,63,24,36,36,24,0,0         :REM p
113 DATA 0,24,36,36,24,63,33,0         :REM q
114 DATA 34,62,18,32,32,16,0,0         :REM r
115 DATA 0,18,42,42,42,36,0,0          :REM s
116 DATA 32,32,252,34,34,4,0,0         :REM t
117 DATA 32,60,34,2,4,62,34,0          :REM u
118 DATA 32,56,36,2,36,56,32,0         :REM v
119 DATA 32,62,2,28,2,62,32,0          :REM w
120 DATA 34,34,20,8,20,34,34,0         :REM x
121 DATA 32,57,37,5,5,62,32,0          :REM y
122 DATA 0,50,38,42,50,38,0,0          :REM z
123 DATA 0,16,108,130,130,130,0,0      :REM {
124 DATA 0,0,0,255,0,0,0,0             :REM |
125 DATA 0,130,130,130,108,16,0,0      :REM }
126 DATA 0,64,128,64,128,0,0,0         :REM ~
```

Abb. C.2: Der Schrifttyp BLIPPO BLACK (Fortsetzung)

```
32 REM SCHRIFTTYPE  GOTHIC
33 DATA 0,0,0,242,0,0,0,0            :REM !
34 DATA 0,224,224,0,224,224,0,0      :REM "
35 DATA 0,40,254,40,254,40,0,0       :REM #
36 DATA 0,36,84,254,84,72,0,0        :REM $
37 DATA 98,100,8,16,38,70,0,0        :REM %
38 DATA 0,12,114,138,100,2,0,0       :REM &
39 DATA 0,0,224,224,0,0,0,0          :REM '
40 DATA 0,0,56,68,130,0,0,0          :REM (
41 DATA 0,0,130,68,56,0,0,0          :REM )
42 DATA 0,84,56,254,56,84,0,0        :REM *
43 DATA 0,16,16,124,16,16,0,0        :REM +
44 DATA 0,0,1,6,0,0,0,0              :REM ,
45 DATA 0,16,16,16,16,16,0,0         :REM -
46 DATA 0,0,6,6,0,0,0,0              :REM .
47 DATA 2,4,8,16,32,64,0,0           :REM /
48 DATA 0,124,138,146,162,124,0,0    :REM 0
49 DATA 0,34,66,254,2,2,0,0          :REM 1
50 DATA 0,70,138,138,146,98,0,0      :REM 2
51 DATA 0,68,130,146,146,108,0,0     :REM 3
52 DATA 8,24,40,72,254,8,0,0         :REM 4
53 DATA 0,244,146,146,146,140,0,0    :REM 5
54 DATA 0,60,82,146,146,12,0,0       :REM 6
55 DATA 0,192,128,158,160,192,0,0    :REM 7
56 DATA 0,108,146,146,146,108,0,0    :REM 8
57 DATA 0,96,146,146,148,120,0,0     :REM 9
58 DATA 0,0,0,36,0,0,0,0             :REM :
59 DATA 0,0,1,38,0,0,0,0             :REM ;
60 DATA 0,16,40,68,130,130,0,0       :REM <
61 DATA 0,40,40,40,40,40,0,0         :REM =
62 DATA 0,130,130,68,40,16,0,0       :REM >
63 DATA 0,64,128,138,144,96,0,0      :REM ?
64 DATA 0,124,146,170,154,114,0,0    :REM @
65 DATA 66,156,168,72,40,28,2,0      :REM A
66 DATA 186,84,146,146,146,108,0,0   :REM B
67 DATA 56,68,186,130,162,68,0,0     :REM C
68 DATA 254,68,130,130,68,56,0,0     :REM D
69 DATA 186,84,146,146,130,68,0,0    :REM E
70 DATA 146,124,144,144,128,64,0,0   :REM F
71 DATA 56,76,148,148,149,95,0,0     :REM G
72 DATA 2,124,144,144,144,78,0,0     :REM H
73 DATA 34,68,68,124,68,68,136,0     :REM I
74 DATA 4,10,2,2,4,120,128,0         :REM J
75 DATA 146,124,16,40,68,130,0,0     :REM K
76 DATA 2,124,130,130,66,2,4,0       :REM L
77 DATA 146,124,128,124,128,124,0,0  :REM M
78 DATA 146,124,64,60,2,124,128,0    :REM N
79 DATA 56,68,186,130,130,68,56,0    :REM O
```

Abb. C.3: Der Schrifttyp GOTHIC

```
 80 DATA 146,124,144,144,144,96,0,0       :REM P
 81 DATA 66,162,198,134,138,74,52,0       :REM Q
 82 DATA 146,124,144,144,152,102,0,0      :REM R
 83 DATA 34,84,84,84,84,84,136,0          :REM S
 84 DATA 88,100,122,66,66,66,132,0        :REM T
 85 DATA 64,128,124,2,2,124,128,0         :REM U
 86 DATA 64,128,120,68,130,132,120,0      :REM V
 87 DATA 128,124,2,124,66,68,56,0         :REM W
 88 DATA 66,132,72,56,36,66,132,0         :REM X
 89 DATA 68,130,114,10,10,114,140,0       :REM Y
 90 DATA 0,134,138,146,162,194,0,0        :REM Z
 91 DATA 0,254,130,130,130,0,0,0          :REM [
 92 DATA 64,32,16,8,4,2,0,0               :REM \
 93 DATA 0,130,130,130,254,0,0,0          :REM ]
 94 DATA 0,32,64,254,64,32,0,0            :REM ↑
 95 DATA 1,1,1,1,1,1,1,0                  :REM _
 96 DATA 0,0,128,64,32,0,0,0              :REM `
 97 DATA 4,42,42,42,28,2,0,0              :REM a
 98 DATA 0,254,20,34,34,28,0,0            :REM b
 99 DATA 0,28,34,34,34,20,0,0             :REM c
100 DATA 0,28,34,34,20,254,0,0            :REM d
101 DATA 0,28,42,42,42,24,0,0             :REM e
102 DATA 16,16,126,144,144,64,0,0         :REM f
103 DATA 0,24,37,37,25,62,0,0             :REM g
104 DATA 0,254,16,32,32,30,0,0            :REM h
105 DATA 0,0,34,190,2,0,0,0               :REM i
106 DATA 0,2,1,1,33,190,0,0               :REM j
107 DATA 0,254,4,8,20,34,0,0              :REM k
108 DATA 0,0,130,254,2,0,0,0              :REM l
109 DATA 32,62,32,62,32,30,0,0            :REM m
110 DATA 0,62,16,32,32,30,0,0             :REM n
111 DATA 0,28,34,34,34,28,0,0             :REM o
112 DATA 0,63,24,36,36,24,0,0             :REM p
113 DATA 0,24,36,36,24,63,0,0             :REM q
114 DATA 0,62,16,32,32,16,0,0             :REM r
115 DATA 0,18,42,42,42,36,0,0             :REM s
116 DATA 32,32,252,34,34,4,0,0            :REM t
117 DATA 0,60,2,2,4,62,0,0                :REM u
118 DATA 0,56,4,2,4,56,0,0                :REM v
119 DATA 0,62,2,28,2,62,0,0               :REM w
120 DATA 0,34,20,8,20,34,0,0              :REM x
121 DATA 0,56,5,5,5,62,0,0                :REM y
122 DATA 0,34,38,42,50,34,0,0             :REM z
123 DATA 0,16,108,130,130,130,0,0         :REM {
124 DATA 0,0,0,255,0,0,0,0                :REM |
125 DATA 0,130,130,130,108,16,0,0         :REM }
126 DATA 0,64,128,64,128,0,0,0            :REM ~
```

Abb. C.3: Der Schrifttyp GOTHIC (Fortsetzung)

```
 80 DATA 254,254,216,216,248,112,0,0   :REM P
 81 DATA 124,254,194,202,196,122,0,0   :REM Q
 82 DATA 254,254,216,220,254,118,0,0   :REM R
 83 DATA 100,246,214,214,222,76,0,0    :REM S
 84 DATA 192,192,254,254,192,192,0,0   :REM T
 85 DATA 252,254,6,6,254,252,0,0       :REM U
 86 DATA 248,252,6,6,252,248,0,0       :REM V
 87 DATA 254,254,4,24,4,254,254,0      :REM W
 88 DATA 198,238,56,56,238,198,0,0     :REM X
 89 DATA 224,240,30,30,240,224,0,0     :REM Y
 90 DATA 198,206,222,246,230,198,0,0   :REM Z
 91 DATA 254,254,198,198,198,0,0,0     :REM [
 92 DATA 192,224,112,56,28,14,6,0      :REM \
 93 DATA 198,198,198,254,254,0,0,0     :REM ]
 94 DATA 32,96,224,224,96,32,0,0       :REM ^
 95 DATA 3,3,3,3,3,3,3,0               :REM _
 96 DATA 0,128,192,96,48,16,0,0        :REM `
 97 DATA 4,46,42,42,62,30,0,0          :REM a
 98 DATA 254,254,34,34,62,28,0,0       :REM b
 99 DATA 28,62,34,34,54,20,0,0         :REM c
100 DATA 28,62,34,34,254,254,0,0       :REM d
101 DATA 28,62,42,42,58,16,0,0         :REM e
102 DATA 16,126,254,144,208,64,0,0     :REM f
103 DATA 24,61,37,37,63,30,0,0         :REM g
104 DATA 254,254,32,32,62,30,0,0       :REM h
105 DATA 0,0,94,94,0,0,0,0             :REM i
106 DATA 6,7,1,1,191,190,0,0           :REM j
107 DATA 254,254,8,28,54,34,0,0        :REM k
108 DATA 0,130,254,254,2,0,0,0         :REM l
109 DATA 62,62,32,62,62,32,30,0        :REM m
110 DATA 62,62,32,32,62,30,0,0         :REM n
111 DATA 28,62,34,34,62,28,0,0         :REM o
112 DATA 63,63,36,36,60,24,0,0         :REM p
113 DATA 24,60,36,36,63,63,0,0         :REM q
114 DATA 62,62,32,32,48,16,0,0         :REM r
115 DATA 16,58,42,42,46,4,0,0          :REM s
116 DATA 32,252,254,34,38,4,0,0        :REM t
117 DATA 60,62,2,2,62,62,0,0           :REM u
118 DATA 56,60,6,6,60,56,0,0           :REM v
119 DATA 60,2,62,62,2,62,62,0          :REM w
120 DATA 34,54,28,28,54,34,0,0         :REM x
121 DATA 56,61,5,5,63,62,0,0           :REM y
122 DATA 34,38,46,58,50,34,0,0         :REM z
123 DATA 16,124,254,198,198,0,0,0      :REM {
124 DATA 0,0,255,255,0,0,0,0           :REM |
125 DATA 0,198,198,254,124,16,0,0      :REM }
126 DATA 0,0,16,32,16,32,0,0           :REM ~
```

Abb. C.4: Der Schrifttyp COLOSSAL (Fortsetzung)

```
32 REM SCHRIFTTYPE  BYTE
33 DATA 0,0,250,250,250,0,0,0          :REM !
34 DATA 0,208,224,0,208,224,0,0        :REM "
35 DATA 68,254,68,68,68,254,68,0       :REM #
36 DATA 0,116,84,214,84,92,0,0         :REM $
37 DATA 226,164,232,16,46,74,142,0     :REM %
38 DATA 0,110,146,138,68,2,0,0         :REM &
39 DATA 0,0,208,208,224,0,0,0          :REM '
40 DATA 0,56,124,198,130,0,0,0         :REM (
41 DATA 0,130,198,124,56,0,0,0         :REM )
42 DATA 68,40,254,40,68,0,0,0          :REM *
43 DATA 0,16,16,124,28,16,0,0          :REM +
44 DATA 0,0,13,13,14,0,0,0             :REM ,
45 DATA 0,24,24,24,24,24,0,0           :REM -
46 DATA 0,0,14,14,14,0,0,0             :REM .
47 DATA 2,4,8,16,32,64,128,0           :REM /
48 DATA 254,142,146,162,254,0,0,0      :REM 0
49 DATA 130,130,254,2,2,0,0,0          :REM 1
50 DATA 158,158,146,146,146,242,0,0    :REM 2
51 DATA 2,146,146,158,254,0,0,0        :REM 3
52 DATA 248,8,8,14,254,8,0,0           :REM 4
53 DATA 246,146,146,158,158,0,0,0      :REM 5
54 DATA 254,158,146,146,222,0,0,0      :REM 6
55 DATA 192,142,158,160,192,0,0,0      :REM 7
56 DATA 254,158,146,146,254,0,0,0      :REM 8
57 DATA 246,146,146,158,254,0,0,0      :REM 9
58 DATA 0,0,108,108,108,0,0,0          :REM :
59 DATA 0,0,109,109,110,0,0,0          :REM ;
60 DATA 16,16,40,68,130,0,0,0          :REM <
61 DATA 0,108,108,108,108,108,0,0      :REM =
62 DATA 130,68,40,16,16,0,0,0          :REM >
63 DATA 0,192,128,154,144,144,240,0    :REM ?
64 DATA 0,0,224,160,160,224,0,0        :REM °
65 DATA 254,174,160,160,224,62,0,0     :REM A
66 DATA 254,174,162,162,226,62,0,0     :REM B
67 DATA 254,158,130,130,130,198,0,0    :REM C
68 DATA 254,158,130,130,130,124,0,0    :REM D
69 DATA 254,174,162,162,130,130,0,0    :REM E
70 DATA 254,174,160,160,160,128,0,0    :REM F
71 DATA 254,142,130,146,146,222,0,0    :REM G
72 DATA 254,46,32,32,32,254,0,0        :REM H
73 DATA 0,0,254,30,0,0,0,0             :REM I
74 DATA 4,2,2,2,30,252,0,0             :REM J
75 DATA 254,30,16,48,72,134,0,0        :REM K
76 DATA 254,30,2,2,2,2,0,0             :REM L
77 DATA 254,142,128,254,128,128,126,0:REM M
78 DATA 254,142,128,124,2,254,0,0      :REM N
79 DATA 254,142,130,130,130,254,0,0    :REM O
```

Abb. C.5: Der Schrifttyp BYTE

```
 80 DATA 254,158,144,144,144,240,0,0   :REM P
 81 DATA 254,142,130,154,132,250,0,0   :REM Q
 82 DATA 254,158,144,144,152,246,0,0   :REM R
 83 DATA 242,146,146,146,158,158,0,0   :REM S
 84 DATA 128,128,254,142,128,128,0,0   :REM T
 85 DATA 254,30,2,2,2,254,0,0          :REM U
 86 DATA 240,56,4,2,4,248,0,0          :REM V
 87 DATA 254,14,2,62,2,2,254,0         :REM W
 88 DATA 134,72,48,48,72,134,0,0       :REM X
 89 DATA 240,16,30,22,16,240,0,0       :REM Y
 90 DATA 134,142,146,162,194,130,0,0   :REM Z
 91 DATA 0,254,130,130,130,0,0,0       :REM [
 92 DATA 128,64,32,16,8,4,2,0          :REM \
 93 DATA 0,0,130,130,130,254,0,0       :REM ]
 94 DATA 8,24,48,96,48,24,8,0          :REM ^
 95 DATA 0,0,0,0,0,0,0,0               :REM '
 96 DATA 0,0,0,0,0,0,0,0               :REM
 97 DATA 0,46,42,42,42,62,0,0          :REM a
 98 DATA 0,254,34,34,34,62,0,0         :REM b
 99 DATA 0,62,34,34,34,34,0,0          :REM c
100 DATA 0,62,34,34,34,254,0,0         :REM d
101 DATA 0,62,42,42,42,58,0,0          :REM e
102 DATA 0,32,254,160,160,128,0,0      :REM f
103 DATA 0,61,37,37,37,63,0,0          :REM g
104 DATA 0,254,32,32,32,62,0,0         :REM h
105 DATA 0,0,0,190,190,0,0,0           :REM i
106 DATA 0,3,1,1,95,0,0,0              :REM j
107 DATA 0,254,8,20,34,0,0,0           :REM k
108 DATA 0,0,0,254,254,0,0,0           :REM l
109 DATA 62,32,32,62,32,32,62,0        :REM m
110 DATA 0,62,32,32,32,62,0,0          :REM n
111 DATA 0,62,34,34,34,62,0,0          :REM o
112 DATA 0,63,36,36,36,60,0,0          :REM p
113 DATA 0,60,36,36,36,63,0,0          :REM q
114 DATA 0,62,32,32,32,48,0,0          :REM r
115 DATA 0,58,42,42,42,46,0,0          :REM s
116 DATA 0,32,126,34,34,34,0,0         :REM t
117 DATA 0,0,62,2,2,62,0,0             :REM u
118 DATA 0,56,4,2,4,56,0,0             :REM v
119 DATA 62,2,2,30,2,2,62,0            :REM w
120 DATA 0,34,20,8,20,34,0,0           :REM x
121 DATA 0,61,5,5,5,63,0,0             :REM y
122 DATA 0,34,38,42,50,34,0,0          :REM z
123 DATA 0,0,0,0,0,0,0,0               :REM
124 DATA 0,0,0,127,0,0,0,0             :REM |
125 DATA 0,0,0,0,0,0,0,0               :REM
126 DATA 0,0,0,0,0,0,0,0               :REM
```

Abb. C.5: Der Schrifttyp BYTE (Fortsetzung)

```
32 REM SCHRIFTTYPE  STOP
33 DATA 0,0,246,246,0,0,0,0           :REM !
34 DATA 0,224,192,0,224,192,0,0       :REM "
35 DATA 36,126,126,0,126,36,0,0       :REM #
36 DATA 36,84,84,214,84,72,0,0        :REM $
37 DATA 227,167,232,23,229,199,0,0    :REM %
38 DATA 12,114,146,146,146,146,0,0    :REM &
39 DATA 0,0,224,192,0,0,0,0           :REM '
40 DATA 0,0,56,68,130,130,0,0         :REM (
41 DATA 130,130,68,56,0,0,0,0         :REM )
42 DATA 0,168,112,216,112,168,0,0     :REM *
43 DATA 0,16,16,124,16,16,0,0         :REM +
44 DATA 0,7,6,0,0,0,0,0               :REM ,
45 DATA 0,16,16,16,16,0,0,0           :REM -
46 DATA 0,6,6,0,0,0,0,0               :REM .
47 DATA 0,6,14,16,224,192,0,0         :REM /
48 DATA 124,138,146,162,194,252,0,0   :REM 0
49 DATA 0,0,64,128,254,0,0,0          :REM 1
50 DATA 138,146,146,146,146,98,0,0    :REM 2
51 DATA 130,130,146,146,210,140,0,0   :REM 3
52 DATA 16,48,80,150,150,16,0,0       :REM 4
53 DATA 146,146,146,146,146,140,0,0   :REM 5
54 DATA 0,12,50,194,18,12,0,0         :REM 6
55 DATA 128,128,134,152,160,128,0,0   :REM 7
56 DATA 108,146,130,130,146,108,0,0   :REM 8
57 DATA 0,96,144,134,152,96,0,0       :REM 9
58 DATA 0,54,54,0,0,0,0,0             :REM :
59 DATA 0,55,54,0,0,0,0,0             :REM ;
60 DATA 16,40,68,16,40,68,0,0         :REM <
61 DATA 0,40,40,40,40,0,0,0           :REM =
62 DATA 68,40,16,68,40,16,0,0         :REM >
63 DATA 0,128,128,138,144,96,0,0      :REM ?
64 DATA 12,18,30,2,60,18,0,0          :REM @
65 DATA 0,14,50,194,50,14,0,0         :REM A
66 DATA 158,146,146,146,146,108,0,0   :REM B
67 DATA 56,68,130,130,130,130,0,0     :REM C
68 DATA 190,130,130,130,68,56,0,0     :REM D
69 DATA 158,146,146,146,146,146,0,0   :REM E
70 DATA 158,144,144,144,144,144,0,0   :REM F
71 DATA 56,68,130,146,146,148,0,0     :REM G
72 DATA 254,16,16,16,0,254,0,0        :REM H
73 DATA 0,0,130,190,130,0,0,0         :REM I
74 DATA 0,2,2,130,188,128,0,0         :REM J
75 DATA 254,0,16,40,68,130,0,0        :REM K
76 DATA 254,2,2,2,2,2,0,0             :REM L
77 DATA 254,128,190,128,128,126,0,0   :REM M
78 DATA 254,128,128,128,128,126,0,0   :REM N
79 DATA 124,130,130,130,130,124,0,0   :REM O
```

Abb. C.6: Der Schrifttyp STOP

```
 80 DATA 158,144,144,144,144,96,0,0     :REM P
 81 DATA 124,130,129,129,129,125,0,0    :REM Q
 82 DATA 144,144,144,144,144,110,0,0    :REM R
 83 DATA 0,2,194,34,18,12,0,0           :REM S
 84 DATA 0,188,130,130,130,130,0,0      :REM T
 85 DATA 252,2,2,2,2,254,0,0            :REM U
 86 DATA 224,24,6,6,24,224,0,0          :REM V
 87 DATA 240,14,192,48,14,240,0,0       :REM W
 88 DATA 0,130,68,56,68,130,0,0         :REM X
 89 DATA 0,224,22,22,22,224,0,0         :REM Y
 90 DATA 130,134,138,146,162,130,0,0    :REM Z
 91 DATA 0,0,255,129,129,129,0,0        :REM [
 92 DATA 192,224,16,8,7,3,0,0           :REM \
 93 DATA 129,129,129,255,0,0,0,0        :REM ]
 94 DATA 0,8,24,40,24,8,0,0             :REM ^
 95 DATA 1,1,1,1,1,1,1,0                :REM _
 96 DATA 0,0,192,224,0,0,0,0            :REM `
 97 DATA 28,34,34,32,32,62,0,0          :REM a
 98 DATA 254,2,34,34,34,28,0,0          :REM b
 99 DATA 28,34,34,34,34,34,0,0          :REM c
100 DATA 28,34,34,34,2,254,0,0          :REM d
101 DATA 28,34,34,42,50,34,0,0          :REM e
102 DATA 0,126,144,144,144,0,0,0        :REM f
103 DATA 25,37,37,33,33,30,0,0          :REM g
104 DATA 254,0,32,32,32,30,0,0          :REM h
105 DATA 0,0,0,190,0,0,0,0              :REM i
106 DATA 0,0,1,1,190,0,0,0              :REM j
107 DATA 254,0,8,8,20,34,0,0            :REM k
108 DATA 0,0,0,254,0,0,0,0              :REM l
109 DATA 62,32,30,32,32,30,0,0          :REM m
110 DATA 62,32,32,32,32,30,0,0          :REM n
111 DATA 28,34,34,34,34,28,0,0          :REM o
112 DATA 63,32,34,34,34,28,0,0          :REM p
113 DATA 28,34,34,34,32,63,0,0          :REM q
114 DATA 0,30,32,32,32,0,0,0            :REM r
115 DATA 18,42,42,42,42,36,0,0          :REM s
116 DATA 0,252,34,34,34,0,0,0           :REM t
117 DATA 60,2,2,2,2,60,0,0              :REM u
118 DATA 0,56,4,2,4,56,0,0              :REM v
119 DATA 60,2,2,60,2,62,0,0             :REM w
120 DATA 0,34,20,8,20,34,0,0            :REM x
121 DATA 57,5,35,34,36,24,0,0           :REM y
122 DATA 34,38,42,42,50,34,0,0          :REM z
123 DATA 72,252,252,0,252,72,0,0        :REM #
124 DATA 0,40,254,40,40,254,40,0        :REM #
125 DATA 0,40,254,40,254,40,0,0         :REM #
126 DATA 0,0,0,0,0,0,0,0                :REM
```

Abb. C.6: Der Schrifttyp STOP (Fortsetzung)

```
32 REM SCHRIFTTYPE  PUDGY
33 DATA 0,0,0,242,0,0,0,0              :REM !
34 DATA 0,224,224,0,224,224,0,0        :REM "
35 DATA 0,40,254,40,254,40,0,0         :REM #
36 DATA 0,36,84,254,84,72,0,0          :REM $
37 DATA 98,100,8,16,38,70,0,0          :REM %
38 DATA 0,12,114,138,100,2,0,0         :REM &
39 DATA 0,0,224,224,0,0,0,0            :REM '
40 DATA 0,0,56,124,254,0,0,0           :REM (
41 DATA 0,0,254,124,56,0,0,0           :REM )
42 DATA 0,84,56,254,56,84,0,0          :REM *
43 DATA 0,16,16,124,16,16,0,0          :REM +
44 DATA 0,0,1,6,0,0,0,0                :REM ,
45 DATA 0,48,48,48,48,48,0,0           :REM -
46 DATA 0,0,6,6,0,0,0,0                :REM .
47 DATA 2,4,8,16,32,64,0,0             :REM /
48 DATA 0,124,138,146,162,124,0,0      :REM 0
49 DATA 0,34,66,254,2,2,0,0            :REM 1
50 DATA 0,70,138,138,146,98,0,0        :REM 2
51 DATA 0,68,130,146,146,108,0,0       :REM 3
52 DATA 8,24,40,72,254,8,0,0           :REM 4
53 DATA 0,244,146,146,146,140,0,0      :REM 5
54 DATA 0,60,82,146,146,12,0,0         :REM 6
55 DATA 0,192,128,158,160,192,0,0      :REM 7
56 DATA 0,108,146,146,146,108,0,0      :REM 8
57 DATA 0,96,146,146,148,120,0,0       :REM 9
58 DATA 0,0,0,36,0,0,0,0               :REM :
59 DATA 0,0,1,38,0,0,0,0               :REM ;
60 DATA 0,16,56,124,254,0,0,0          :REM <
61 DATA 0,108,108,108,108,108,0,0      :REM =
62 DATA 0,0,254,124,56,16,0,0          :REM >
63 DATA 0,64,128,138,144,96,0,0        :REM ?
64 DATA 0,124,146,170,154,114,0,0      :REM @
65 DATA 0,126,254,240,254,126,0,0      :REM A
66 DATA 0,254,254,254,254,108,0,0      :REM B
67 DATA 0,124,254,254,198,68,0,0       :REM C
68 DATA 0,254,254,254,254,124,0,0      :REM D
69 DATA 0,254,254,254,214,198,0,0      :REM E
70 DATA 0,254,254,254,208,192,0,0      :REM F
71 DATA 0,124,254,254,198,92,0,0       :REM G
72 DATA 0,254,254,56,254,254,0,0       :REM H
73 DATA 0,198,198,254,254,198,0,0      :REM I
74 DATA 4,6,6,254,252,128,0,0          :REM J
75 DATA 0,254,254,238,198,130,0,0      :REM K
76 DATA 0,254,254,6,6,6,0,0            :REM L
77 DATA 0,254,254,112,254,254,0,0      :REM M
78 DATA 0,254,126,56,252,254,0,0       :REM N
79 DATA 0,124,254,254,254,124,0,0      :REM O
```

Abb. C.7: Der Schrifttyp PUDGY

```
80 DATA 0,254,254,240,240,96,0,0      :REM P
81 DATA 0,124,238,246,250,124,0,0     :REM Q
82 DATA 0,254,254,254,246,98,0,0      :REM R
83 DATA 0,100,246,254,222,76,0,0      :REM S
84 DATA 0,192,192,254,254,192,0,0     :REM T
85 DATA 0,252,254,254,254,252,0,0     :REM U
86 DATA 0,248,252,254,252,248,0,0     :REM V
87 DATA 0,254,30,30,30,254,0,0        :REM W
88 DATA 0,198,238,254,238,198,0,0     :REM X
89 DATA 0,224,240,254,240,224,0,0     :REM Y
90 DATA 0,198,206,222,246,230,0,0     :REM Z
91 DATA 0,254,130,130,130,0,0,0       :REM [
92 DATA 64,32,16,8,4,2,0,0            :REM \
93 DATA 0,130,130,130,254,0,0,0       :REM ]
94 DATA 0,32,64,254,64,32,0,0         :REM ↑
95 DATA 1,1,1,1,1,1,1,0               :REM _
96 DATA 0,0,128,64,32,0,0,0           :REM `
97 DATA 4,46,46,46,28,2,0,0           :REM a
98 DATA 0,254,30,62,62,28,0,0         :REM b
99 DATA 0,28,62,62,54,20,0,0          :REM c
100 DATA 0,28,62,62,30,254,0,0        :REM d
101 DATA 0,28,58,58,58,24,0,0         :REM e
102 DATA 0,16,126,208,208,64,0,0      :REM f
103 DATA 0,24,61,61,57,62,0,0         :REM g
104 DATA 0,254,30,48,62,30,0,0        :REM h
105 DATA 0,0,190,190,0,0,0,0          :REM i
106 DATA 0,2,3,3,191,190,0,0          :REM j
107 DATA 0,254,30,30,54,34,0,0        :REM k
108 DATA 0,0,254,254,0,0,0,0          :REM l
109 DATA 0,62,48,56,48,30,0,0         :REM m
110 DATA 0,62,62,48,62,30,0,0         :REM n
111 DATA 0,28,62,62,62,28,0,0         :REM o
112 DATA 0,63,56,60,60,24,0,0         :REM p
113 DATA 0,24,60,60,56,63,0,0         :REM q
114 DATA 0,62,48,48,48,16,0,0         :REM r
115 DATA 0,18,58,62,46,36,0,0         :REM s
116 DATA 0,32,252,254,38,36,0,0       :REM t
117 DATA 0,60,62,6,62,62,0,0          :REM u
118 DATA 0,56,4,2,4,56,0,0            :REM v
119 DATA 0,62,6,14,6,62,0,0           :REM w
120 DATA 0,34,54,62,54,34,0,0         :REM x
121 DATA 0,56,61,13,61,62,0,0         :REM y
122 DATA 0,50,54,62,54,38,0,0         :REM z
123 DATA 0,16,108,130,130,130,0,0     :REM {
124 DATA 0,0,0,255,0,0,0,0            :REM |
125 DATA 0,130,130,130,108,16,0,0     :REM }
126 DATA 0,64,128,64,128,0,0,0        :REM
```

Abb. C.7: Der Schrifttyp PUDGY (Fortsetzung)

```
32 REM SCHRIFTTYPE  PINOCCHIO
33 DATA 0,0,246,246,0,0,0,0              :REM !
34 DATA 0,192,224,0,192,224,0,0          :REM "
35 DATA 0,40,254,40,254,40,0,0           :REM #
36 DATA 102,242,178,255,158,204,0,0      :REM $
37 DATA 196,200,16,32,76,140,0,0         :REM %
38 DATA 238,254,130,222,222,24,0,0       :REM &
39 DATA 0,192,224,0,0,0,0,0              :REM '
40 DATA 0,0,56,68,130,0,0,0              :REM (
41 DATA 0,0,130,68,56,0,0,0              :REM )
42 DATA 0,32,112,216,112,32,0,0          :REM *
43 DATA 0,16,16,124,16,16,0,0            :REM +
44 DATA 0,0,1,6,0,0,0,0                  :REM ,
45 DATA 0,16,16,16,16,16,0,0             :REM -
46 DATA 0,0,6,6,0,0,0,0                  :REM .
47 DATA 2,4,8,16,32,64,0,0               :REM /
48 DATA 0,254,186,130,186,254,0,0        :REM 0
49 DATA 0,0,222,158,254,0,0,0            :REM 1
50 DATA 102,206,158,190,246,102,0,0      :REM 2
51 DATA 198,198,146,146,254,254,0,0      :REM 3
52 DATA 248,248,218,30,254,24,0,0        :REM 4
53 DATA 246,246,178,178,190,156,0,0      :REM 5
54 DATA 124,254,146,146,222,204,0,0      :REM 6
55 DATA 224,198,142,158,248,224,0,0      :REM 7
56 DATA 238,254,186,186,254,238,0,0      :REM 8
57 DATA 102,246,146,146,254,254,0,0      :REM 9
58 DATA 0,54,54,0,0,0,0,0                :REM :
59 DATA 0,54,55,0,0,0,0,0                :REM ;
60 DATA 16,40,68,16,40,68,0,0            :REM <
61 DATA 0,40,40,40,40,40,0,0             :REM =
62 DATA 68,40,16,68,40,16,0,0            :REM >
63 DATA 0,192,219,155,248,248,0,0        :REM ?
64 DATA 0,124,146,170,154,114,0,0        :REM @
65 DATA 126,214,208,208,214,126,0,0      :REM A
66 DATA 254,254,130,186,254,108,0,0      :REM B
67 DATA 124,254,130,130,238,108,0,0      :REM C
68 DATA 254,254,186,130,254,124,0,0      :REM D
69 DATA 254,254,146,146,214,214,0,0      :REM E
70 DATA 254,254,150,146,208,208,0,0      :REM F
71 DATA 124,254,130,138,238,108,0,0      :REM G
72 DATA 254,254,22,208,254,254,0,0       :REM H
73 DATA 0,0,254,254,254,0,0,0            :REM I
74 DATA 28,14,198,242,254,252,0,0        :REM J
75 DATA 254,254,214,16,254,238,0,0       :REM K
76 DATA 254,254,242,198,14,30,0,0        :REM L
77 DATA 254,198,240,240,198,254,0,0      :REM M
78 DATA 254,246,50,152,222,254,0,0       :REM N
79 DATA 0,124,222,198,246,124,0,0        :REM O
```

Abb. C.8: Der Schrifttyp PINOCCHIO

```
 80 DATA 254,254,134,176,240,96,0,0    :REM P
 81 DATA 124,222,198,246,126,6,0,0     :REM Q
 82 DATA 254,254,150,144,254,110,0,0   :REM R
 83 DATA 102,242,186,186,158,204,0,0   :REM S
 84 DATA 242,198,254,254,198,242,0,0   :REM T
 85 DATA 252,254,2,250,254,252,0,0     :REM U
 86 DATA 254,254,6,222,240,192,0,0     :REM V
 87 DATA 254,198,28,28,198,254,0,0     :REM W
 88 DATA 238,254,56,56,254,238,0,0     :REM X
 89 DATA 246,246,22,210,254,254,0,0    :REM Y
 90 DATA 226,142,190,250,226,142,0,0   :REM Z
 91 DATA 0,254,130,130,130,0,0,0       :REM [
 92 DATA 64,32,16,8,4,2,0,0            :REM \
 93 DATA 0,130,130,130,254,0,0,0       :REM ]
 94 DATA 0,32,64,254,64,32,0,0         :REM ↑
 95 DATA 1,1,1,1,1,1,1,0               :REM _
 96 DATA 0,0,128,64,32,0,0,0           :REM `
 97 DATA 110,110,74,74,126,126,0,0     :REM a
 98 DATA 254,254,234,34,62,28,0,0      :REM b
 99 DATA 60,126,66,66,102,102,0,0      :REM c
100 DATA 28,62,34,42,254,254,0,0       :REM d
101 DATA 60,126,82,82,86,118,0,0       :REM e
102 DATA 126,254,254,136,224,96,0,0    :REM f
103 DATA 51,123,73,73,127,63,0,0       :REM g
104 DATA 254,254,208,16,62,62,0,0      :REM h
105 DATA 0,0,190,190,190,0,0,0         :REM i
106 DATA 6,7,1,191,191,190,0,0         :REM j
107 DATA 254,254,214,16,62,38,0,0      :REM k
108 DATA 0,0,254,254,254,0,0,0         :REM l
109 DATA 0,126,102,112,102,126,0,0     :REM m
110 DATA 126,126,102,96,102,62,0,0     :REM n
111 DATA 0,60,110,102,118,60,0,0       :REM o
112 DATA 63,63,36,38,62,28,0,0         :REM p
113 DATA 28,62,38,36,63,63,0,0         :REM q
114 DATA 126,126,126,64,96,96,0,0      :REM r
115 DATA 102,114,122,94,78,102,0,0     :REM s
116 DATA 0,32,252,254,254,46,0,0       :REM t
117 DATA 126,126,6,118,126,126,0,0     :REM u
118 DATA 126,126,14,30,120,96,0,0      :REM v
119 DATA 0,126,102,14,102,126,0,0      :REM w
120 DATA 0,110,126,16,126,110,0,0      :REM x
121 DATA 123,123,11,105,127,127,0,0    :REM y
122 DATA 102,78,94,122,114,102,0,0     :REM z
123 DATA 16,48,80,144,80,48,16,0       :REM ^
124 DATA 0,0,0,0,0,0,0,0               :REM
125 DATA 0,0,0,0,0,0,0,0               :REM
126 DATA 0,0,0,0,0,0,0,0               :REM
```

Abb. C.8: Der Schrifttyp PINOCCHIO (Fortsetzung)

```
32 REM SCHRIFTTYPE  OUTLINE
33 DATA 0,0,0,244,244,244,0,0        :REM !
34 DATA 0,208,224,0,208,224,0,0      :REM "
35 DATA 0,40,124,40,40,124,40,0      :REM #
36 DATA 0,116,84,214,84,92,0,0       :REM $
37 DATA 66,164,72,16,36,74,132,0     :REM
38 DATA 0,108,210,138,68,10,0,0      :REM &
39 DATA 0,0,0,208,224,0,0,0          :REM '
40 DATA 0,56,124,254,198,130,0,0     :REM (
41 DATA 0,130,198,254,124,56,0,0     :REM )
42 DATA 0,168,112,80,112,168,0,0     :REM *
43 DATA 0,16,16,124,16,16,0,0        :REM +
44 DATA 0,0,0,13,14,0,0,0            :REM ,
45 DATA 0,24,24,24,24,24,24,0        :REM -
46 DATA 0,0,0,12,12,0,0,0            :REM .
47 DATA 0,12,28,56,112,224,192,0     :REM /
48 DATA 124,254,154,178,254,124,0,0  :REM 0
49 DATA 34,98,254,254,2,2,0,0        :REM 1
50 DATA 66,198,142,154,242,98,0,0    :REM 2
51 DATA 68,198,146,146,254,108,0,0   :REM 3
52 DATA 24,56,104,200,254,254,0,0    :REM 4
53 DATA 244,246,146,146,158,140,0,0  :REM 5
54 DATA 60,126,210,146,158,12,0,0    :REM 6
55 DATA 192,192,158,190,224,192,0,0  :REM 7
56 DATA 108,254,146,146,254,108,0,0  :REM 8
57 DATA 96,240,146,150,252,120,0,0   :REM 9
58 DATA 0,0,0,108,108,0,0,0          :REM :
59 DATA 0,0,0,109,110,0,0,0          :REM ;
60 DATA 0,16,56,124,254,0,0,0        :REM <
61 DATA 0,40,40,40,40,40,40,0        :REM =
62 DATA 0,0,254,124,56,16,0,0        :REM >
63 DATA 0,96,192,202,218,240,96,0    :REM ?
64 DATA 28,63,63,94,191,63,28,0      :REM @
65 DATA 127,193,183,180,183,193,127,0:REM A
66 DATA 255,129,173,173,173,211,110,0:REM B
67 DATA 126,195,189,165,189,219,126,0:REM C
68 DATA 255,129,189,189,219,102,60,0 :REM D
69 DATA 126,195,173,173,189,219,126,0:REM E
70 DATA 127,193,175,168,184,208,112,0:REM F
71 DATA 126,195,189,165,173,235,110,0:REM G
72 DATA 255,129,239,40,239,129,255,0 :REM H
73 DATA 231,165,189,129,189,165,231,0:REM I
74 DATA 14,11,13,5,253,131,254,0     :REM J
75 DATA 255,129,231,36,90,165,195,0  :REM K
76 DATA 255,129,253,5,5,5,7,0        :REM L
77 DATA 255,129,191,136,191,129,255,0:REM M
78 DATA 255,129,223,40,247,129,255,0 :REM N
79 DATA 126,195,189,165,189,195,126,0:REM O
```

Abb. C.9: Der Schrifttyp OUTLINE

```
 80 DATA 255,129,183,180,180,204,120,0:REM P
 81 DATA 126,195,189,181,185,195,126,0:REM Q
 82 DATA 255,129,183,180,178,205,123,0:REM R
 83 DATA 119,221,173,173,173,179,238,0:REM S
 84 DATA 224,160,191,129,191,160,224,0:REM T
 85 DATA 254,131,253,5,253,131,254,0   :REM U
 86 DATA 248,132,250,5,250,132,248,0   :REM V
 87 DATA 255,129,253,17,253,129,255,0  :REM W
 88 DATA 195,165,90,36,90,165,195,0    :REM X
 89 DATA 224,144,239,17,239,144,224,0  :REM Y
 90 DATA 227,165,169,181,173,149,231,0:REM Z
 91 DATA 0,252,252,204,204,0,0,0        :REM [
 92 DATA 0,192,224,112,56,28,12,0       :REM \
 93 DATA 0,0,0,204,204,252,252,0        :REM ]
 94 DATA 16,48,112,240,112,48,16,0      :REM ^
 95 DATA 2,2,2,2,2,2,2,0                :REM _
 96 DATA 0,0,0,224,208,0,0,0            :REM `
 97 DATA 4,46,42,42,62,30,0,0           :REM a
 98 DATA 254,254,34,34,62,28,0,0        :REM b
 99 DATA 28,62,34,34,54,20,0,0          :REM c
100 DATA 28,62,34,34,254,254,0,0        :REM d
101 DATA 28,62,42,42,58,24,0,0          :REM e
102 DATA 16,126,254,144,208,64,0,0      :REM f
103 DATA 24,61,37,37,63,30,0,0          :REM g
104 DATA 254,254,32,32,62,30,0,0        :REM h
105 DATA 0,0,190,190,0,0,0,0            :REM i
106 DATA 2,3,1,1,191,190,0,0            :REM j
107 DATA 254,254,8,28,54,34,0,0         :REM k
108 DATA 0,0,254,254,0,0,0,0            :REM l
109 DATA 30,62,32,56,32,62,30,0         :REM m
110 DATA 30,62,32,32,62,30,0,0          :REM n
111 DATA 28,62,34,34,62,28,0,0          :REM o
112 DATA 31,63,36,36,60,24,0,0          :REM p
113 DATA 24,60,36,36,63,31,0,0          :REM q
114 DATA 30,62,32,32,48,16,0,0          :REM r
115 DATA 18,58,42,42,46,36,0,0          :REM s
116 DATA 32,252,254,34,38,4,0,0         :REM t
117 DATA 60,62,2,2,62,60,0,0            :REM u
118 DATA 56,60,6,6,60,56,0,0            :REM v
119 DATA 60,62,2,14,2,62,60,0           :REM w
120 DATA 34,54,28,28,54,34,0,0          :REM x
121 DATA 56,61,5,5,63,62,0,0            :REM y
122 DATA 34,38,46,58,50,34,0,0          :REM z
123 DATA 0,0,16,124,254,130,130,0       :REM {
124 DATA 0,0,0,255,255,0,0,0            :REM |
125 DATA 0,130,130,254,124,16,0,0       :REM }
126 DATA 0,0,0,0,0,0,0,0                :REM
```

Abb. C.9: Der Schrifttyp OUTLINE (Fortsetzung)

Anhang D

Alphabetische Liste der BASIC-Schlüsselwörter einschließlich Kurzkommentaren

Anweisung: @-Symbol
Beispiel: PRINT@10,"HALLO"

Das @-Symbol ist eine Funktion, die ausschließlich in Verbindung mit der PRINT-Anweisung verwendet wird. Durch das Symbol wird bestimmt, daß die Ausgabe eines Textes auf dem Bildschirm ab einer bestimmten Bildschirmposition erfolgen soll. Zulässige Werte zur Kennzeichnung der Bildschirmposition liegen zwischen 0 und 959, wobei der Wert 0 die linke obere Bildschirmecke kennzeichnet, 959 die rechte untere (s. a. BASIC-Handbuch S. 126).

Anweisung: ABS(W)
Beispiel: X=ABS(–5)

ABS ist eine numerische Funktion, die den Absolutbetrag der im Argument vereinbarten Zahl liefert. In unserem Beispiel wird der Variablen X der Wert 5 zugewiesen.

Anweisung: AND
Beispiel: IF (X=2) AND (Y=4) THEN PRINT"X=2,Y=4"

AND ist ein logischer Operator.

Er kann verwendet werden, um einen zusammengesetzten Ausdruck innerhalb einer IF/THEN-Anweisung zu bilden und dessen Wahrheitsgehalt zu prüfen. Im Beispiel wird nur dann der Text X=2,Y=4 ausgegeben, wenn die Variable X den Wert 2 und die Variable Y den Wert 4 hat (siehe dazu auch das Kapitel „Operatoren").

Anweisung: ASC(W)
Beispiel: S$="A" : X=ASC(S$)

ASC ist eine Zeichenkettenfunktion. Sie ermittelt den ASCII-Code des ersten im Argument angegebenen Textzeichens.

Anweisung: ATN(W)
Beispiel: X=ATN(–8)

ATN ist eine numerische Funktion. Sie liefert den Arcustangens des in der Klammer vorgegebenen Arguments. (Der Arcustangens einer

Zahl X ist definiert als Winkel, dessen Tangens gleich x ist.) Das Ergebnis einer ATN-Funktion wird im Bogenmaß angegeben. Die Umrechnung eines Winkels in Grad erhalten Sie, wenn Sie das Ergebnis der ATN-Funktion mit (180/3.1415927) multiplizieren.

Anweisung: BGRD
Beispiel: BGRD

Mit BGRD wird im Grafik-Modus auf der FGR-Bildschirmebene die Farbe Magenta als Hintergrundfarbe definiert.

Anweisung: CALL n
Beispiel: CALL 01C9

Mit CALL n wird ein Maschinenunterprogramm aufgerufen, das sich im Rechnerspeicher befindet und die Startadresse n besitzt. Im Beispiel wird das Unterprogramm ausgeführt, zu dem der Rechner verzweigt, wenn er die Anweisung CLS vorfindet.

Anweisung: CDBL
Beispiel: PRINT CDBL (22/7)

Durch die CDBL-Funktion wird der Rechner angewiesen, das Ergebnis einer Berechnung in doppelter Genauigkeit auszugeben oder weiterzuverarbeiten.

Anweisung: CHAR n
Beispiel: CHAR 1

Durch CHAR wird einer der 4 Zeichensätze des Colour-Genie angewählt. Das Argument n muß eine ganze Zahl zwischen 1 und 4 sein.

Anweisung: CHR$(W)
Beispiel: PRINT CHR$(230)

CHR$ ist eine Zeichenkettenfunktion. CHR$ liefert als Ergebnis das Zeichen, das dem ASCII-Code des Arguments entspricht. Der Wert des Arguments muß eine ganze Zahl zwischen 0 und 255 sein.

Anweisung: CINT(W)
Beispiel: PRINT CINT(−1.5)

CINT (s. a. INT(W)) liefert den ganzzahligen Wert einer Zahl. Bei positiven Zahlen sieht das so aus, daß einfach der Bruchteil der Zahl gestrichen wird. Ist das Argument W der CINT-Funktion negativ, so liefert CINT die nächstniedrigere ganze Zahl.

Der einzige Unterschied zwischen CINT und INT ist der, daß für CINT als Argument nur Werte zwischen −32768 und +32767 zulässig sind. Für die INT-Funktion gilt diese Beschränkung nicht.

Anweisung: CIRCLE x1,y1,r
Beispiel: CIRCLE 40,20,10

CIRCLE ist ein Grafikbefehl, der das Zeichnen eines Kreises bewirkt. Kreise können nur auf der FGR-Bildschirmebene dargestellt werden (s. a. FGR). Vor der CIRCLE-Anweisung muß eine Bildschirmhintergrundfarbe und eine Malfarbe vorgegeben werden (s. a. FCLS n und FCOLOUR n). Die Argumente x1 und y1 kennzeichnen den Kreismittelpunkt (X- und Y-Koordinate). Für x1 sind Werte zwischen 0 und 159, für y1 Werte zwischen 0 und 102 zulässig.

Anweisung: CLEAR n
Beispiel: CLEAR 2000

CLEAR löscht zuerst alle Variablen. Dann wird so viel Speicherplatz (in Bytes) für die Daten von Zeichenkettenvariablen reserviert, wie im Argument n vorgegeben ist.

Anweisung: CLOSE n (nur Disk-BASIC)
Beispiel: CLOSE 1

Mit der Anweisung CLOSE wird eine auf Diskette angelegte und mit OPEN eröffnete Datei geschlossen. Das Argument n kennzeichnet die Kanalnummer, unter der die Datei eröffnet wurde. Wird kein Argument angegeben, dann werden alle eröffneten Dateien geschlossen.

Anweisung: CLS
Beispiel: CLS

Durch die CLS-Anweisung wird der LGR-Bildschirm (s. a. LGR) gelöscht und der Cursor in die linke obere Bildschirmecke positioniert.

Anweisung: COLOUR n
Beispiel: COLOUR 1

Durch die COLOUR-Anweisung wird die Vordergrundfarbe (das ist die Farbe, in der alle Folgezeichen auf dem LGR-Bildschirm ausgegeben werden sollen) angewählt. Das Argument n kann Werte zwischen 1 und 16 annehmen.

Anweisung: CONT
Beispiel: CONT

Mit der CONT-Anweisung wird ein ablaufendes und durch Betätigung der BREAK-Taste oder durch eine STOP-Anweisung unterbrochenes Programm ab der Stelle fortgeführt, an der die Unterbrechung erfolgte. Die CONT-Anweisung wird nur dann vom Rechner ausgeführt, wenn zwischenzeitlich keine Veränderungen am Programm vorgenommen wurden.

Anweisung: COS(W)
Beispiel: PRINT COS(3.1415927)

Die Cosinus-Funktion liefert den Cosinus eines Wikels W. Der Winkel muß im Bogenmaß angegeben werden. Soll die Eingabe im Gradmaß erfolgen, lautet die Eingabe COS(W * 0.01745329) oder COS(W / 180 * 3.1415927).

Anweisung: CPOINT(x1,y1)
Beispiel: PRINT CPOINT(40,30)

CPOINT ist eine Grafikanweisung. CPOINT prüft, ob sich an der durch (x1,y1) vorgegebenen Bildschirmstelle auf der FGR-Bildschirmebene ein Bildpunkt befindet oder nicht. Ist dies der Fall, liefert CPOINT den Wert −1. Befindet sich an der betreffenden Bildschirmstelle kein Bildpunkt, liefert CPOINT den Wert 0.

Anweisung: DATA
Beispiel: DATA 10,4,"ANTON"

Die DATA-Anweisung enthält Datenelemente, die durch einen READ-Befehl in ein Programm übernommen werden (s. a. RESTORE READ).

Anweisung: DEFDBL n
Beispiel: DEFDBL A,B−D

Durch DEFDBL werden alle in einem Programm erscheinenden Variablen, die den Anfangsbuchstaben n haben, als Variable mit doppelter Genauigkeit definiert. Variablen doppelter Genauigkeit werden vom Rechner mit 16 signifikanten Ziffern gespeichert.

Anweisung: DEFFN f(x) (nur unter Disk-BASIC oder nach Laden des Maschinenprogramms aus dem Anhang A verfügbar).
Beispiel: DEFFN F(X)=SQR(X)

DEF FN definiert eine Funktion, die im Programm durch FN name(x) aufgerufen werden kann.

Anweisung: DEFINT n
Beispiel: DEFINT A,B−D

Durch DEFINT werden alle in einem Programm erscheinenden Variablen, die den Anfangsbuchstaben n haben, als ganzzahlig definiert (s. a. DEFDBL).

Anweisung: DEFSNG n
Beispiel: DEFSNG A,B−D

Durch DEFSNG werden alle in einem Programm erscheinenden Variablen, die den Anfangsbuchstaben n haben, als Variablen mit einfacher Genauigkeit definiert (s. a. DEFDBL).

Anweisung: DEFSTR n
Beispiel: DEFSTR A,B–D

Durch DEFSTR werden alle in einem Programm erscheinenden Variablen, die den Anfangsbuchstaben n haben, als Zeichenkettenvariablen definiert (s. a. DEFDBL).

Anweisung: DELETE n
Beispiel: DELETE 10,20–60

Durch die DELETE-Anweisung werden einzelne Zeilen oder ganze Programmteile gelöscht.

Anweisung: DIM
Beispiel: DIM A(20)

Durch die DIM-Anweisung wird die Größe eines Variablenfeldes definiert. Eine DIM-Anweisung muß nur dann angegeben werden, wenn die Anzahl der Elemente eines Feldes größer als 10 sein soll. Bei Zeichenkettenfeldern muß vor der Dimensionierung durch eine CLEAR-Anweisung ausreichend Platz zur Verfügung gestellt werden.

Anweisung: EDIT n
Beispiel: EDIT 10

Die EDIT-Anweisung ruft den Zeileneditor zur Korrektur von Programmzeilen auf.

Anweisung: END
Beispiel: IF W=100 THEN END

Ende-Statement eines BASIC-Programms. Es kann weggelassen werden, wenn nachfolgend keine Unterprogramme oder Programmsegmente mehr vereinbart sind.

Anweisung: ERL
Beispiel: PRINT ERL

Die ERL-Anweisung gibt die Programmzeile an, in der der Rechner auf einen Fehler stieß.

Anweisung: ERR
Beispiel: PRINT ERR/2+1

Die ERR-Anweisung sollte ausschließlich wie im Beispiel verwendet werden. Sie gibt dann die Fehlercodenummer an, die dem aufgetretenen Feh-

ler entspricht. Die auf Seite 47 des BASIC-Handbuchs aufgeführten Fehlermeldungen sind dort zwar ohne Fehlercodenummer aufgeführt, stehen diesbezüglich jedoch in der richtigen Reihenfolge.

Anweisung: ERROR n
Beispiel: ERROR 10

Die ERROR-Anweisung dient zur Simulation eines Fehlers. Wird die ERROR-Anweisung eingesetzt, so verhält sich der Rechner genauso, als wäre der Fehler tatsächlich aufgetreten.

Anweisung: EXP(W)
Beispiel: PRINT EXP(4)

Die EXP-Funktion ist eine numerische Funktion. Sie berechnet die Exponentialfunktion des Arguments X.

Anweisung: FCLS n
Beispiel: FCLS 1

Die FCLS-Anweisung ist eine Grafikfunktion. Durch sie wird die Hintergrundfarbe des hochauflösenden Bildschirms ausgewählt. Zulässig für das Argument n sind Werte zwischen 1 und 4.

Anweisung: FCOLOUR n
Beispiel: FCOLOUR 2

Die FCOLOUR-Anweisung ist eine Grafikfunktion. Durch sie wird die Vordergrundfarbe des hochauflösenden Bildschirms definiert. Zulässig für das Argument n sind Werte zwischen 1 und 4. Vorder- und Hintergrundfarbe des FGR-Bildschirms dürfen nicht identisch sein (s. a. FCLS n).

Anweisung: FGR
Beispiel: FGR

Durch die FGR-Anweisung wird auf den höchstauflösenden Grafikbildschirm umgeschaltet. Diese Umschaltung kann auch über die Tastatur durch Betätigung der Tasten <CTRL> und <MOD SEL> erfolgen.

Anweisung: FILL n
Beispiel: –

FILL n wurde durch FCLS n ersetzt.

Anweisung: FIX
Beispiel: PRINT FIX(−2.5)

FIX ist eine numerische Funktion. Sie bildet im Gegensatz zu INT den ganzzahligen Wert einer Zahl nicht durch Abrunden, sondern durch Abschneiden der Nachkommastellen. FIX(−2.5) ist −2, INT(−2.5) jedoch −3!

Anweisung: FKEY
Beispiel: FKEY1="PRINT@"

Durch die FKEY-Anweisung können den 8 Funktionstasten (<F1> bis <F8>) neue Werte zugewiesen werden. Diese werden dann bei Betätigung der jeweiligen Funktionstaste auf dem Bildschirm ausgegeben. Wenn Sie in einem Programm abfragen wollen, ob eine bestimmte Funktionstaste betätigt wurde, müssen Sie die Speicheradresse F808 abfragen. Bei Betätigung von <F1> steht an dieser Adresse eine 16, bei <F2> eine 32, bei <F3> eine 64 und bei <F4> eine 128. Wurde zusätzlich noch die <Shift>-Taste betätigt, enthält die Speicheradresse F880 eine 1.

Anweisung: FOR ... NEXT ... STEP
Beispiel: FOR X=1 TO 5 STEP .5

Schleifenvereinbarung.

Anweisung: GET k,n (nur unter Disk-BASIC verfügbar)
Beispiel: GET 1,4

GET dient zum Lesen eines Satzes aus einer Direktzugriffsdatei. Das Argument k ist die Kanalnummer, unter der die Datei eröffnet wurde. n ist die Satznummer, die gelesen werden soll.

Anweisung: GOSUB n
Beispiel: GOSUB 1000

Durch die GOSUB-Anweisung wird ein Unterprogramm angesprungen.

Anweisung: GOTO n
Beispiel: GOTO 100

Durch die GOTO-Anweisung erfolgt eine unbedingte Verzweigung zu der durch n vorgegebenen Zeilennummer.

Anweisung: IF ... THEN ... ELSE
Beispiel: IF X=10 THEN 20 ELSE 30

Abfrage einer logischen Bedingung mit alternativer Ausführung oder bedingten Sprüngen.

Anweisung: INKEY$
Beispiel: Y$=INKEY$

INKEY$ ist eine Zeichenkettenfunktion. Trifft der Rechner auf ein INKEY$, fragt er für Sekundenbruchteile die Tastatur ab, ob dort eine Taste betätigt wurde. Ist dies der Fall, wird der ASCII-Wert der betätigten Taste an die Zeichenkettenvariable übergeben.

Anweisung: INPUT w
Beispiel: INPUT A

Allgemeiner Befehl zur Eingabe von Daten.

Anweisung: INSTR(n,w1,w2)
Beispiel: X=INSTR("3,"AABBBC","B")

INSTR ist eine Zeichenkettenfunktion. Sie testet eine Zeichenkette auf das Vorhandensein eines vorgegebenen Textelements. Im Beispiel wird die Stelle in der Zeichenkette AABBBC gesucht, an der der Buchstabe B zum drittenmal auftritt.

Anweisung: INT
Beispiel: PRINT INT(−2.5)

INT ist eine numerische Funktion. Sie bildet aus einer gebrochenen Zahl einen ganzzahligen Wert mit vorzeichenabhängiger Rundung. (Vgl. auch CINT, FIX).

Anweisung: JOY(n)
Beispiel: PRINT JOY(1)

JOY ermittelt einen der Potentiometerstellung der Joysticks proportionalen Zahlenwert zwischen 0 und 255.

Anweisung: KEYPAD(n)
Beispiel: PRINT KEYPAD(1)

KEYPAD ermöglicht ein indiziertes Ansprechen der Joystick-Tastaturen.

Anweisung: LEFT$ (s$,n)
Beispiel: PRINT LEFT$("123",2)

LEFT$ ist eine Zeichenkettenfunktion. Eine Zeichenkette S wird linksbündig nach dem n-ten Zeichen „abgeschnitten".

Anweisung: LET
Beispiel: LET X=4

Durch die LET-Anweisung wird die Wertzuweisung an eine Variable besonders betont.

Anweisung: LSET (Disk-BASIC-Anweisung)
Beispiel: LSET A$=B$

Durch die LSET-Anweisung wird eine Zeichenkette linksbündig in einer anderen abgelegt.

Anweisung: LEN(s$)
Beispiel: L=LEN(S$)

LEN ist eine Zeichenkettenfunktion. Das Ergebnis ist die Anzahl der im Argument enthaltenen Zeichen.

Anweisung: LGR
Beispiel: LGR

Durch die LGR-Anweisung wird vom FGR-Bildschirm für hochauflösende Grafik auf den Textbildschirm umgeschaltet.

Anweisung: LINE INPUT (Nur unter Disk-BASIC verfügbar)
Beispiel: LINE INPUT A$

LINE INPUT ist ein Befehl zur Eingabe beliebiger Tastaturzeichen (im Gegensatz zu INPUT können hierzu auch Zeichen wie : und , gehören).

Anweisung: LIST
Beispiel: LIST 10–50

LIST erstellt ein Programlisting auf dem Bildschirm.

Anweisung: LLIST
Beispiel: LLIST 10–50

LLIST gibt ein Programmlisting auf einem angeschlossenen Drucker aus.

Anweisung: MEM
Beispiel: PRINT MEM

MEM ermittelt den für den Anwender verbleibenden Speicherumfang in Bytes.

Anweisung: MID$ (S$,x,n)
Beispiel: A$=MID$("ANFANG",3,2)

MID$ ist eine Zeichenkettenfunktion zur Trennung einzelner oder mehrerer Zeichen aus Textelementen.

Anweisung: NAME
Beispiel: NAME

NAME ist eine Anweisung, die ähnlich wie USR vom Anwender über bestimmte Speicheradressen mit Daten versehen wird. Die Werte, die in

diese Speicheradressen geladen werden, werden dann bei Aufruf von NAME vom Rechner als Startadresse eines Unterprogramms interpretiert. NAME wird über die Adressen 16782 und 16783 mit Daten versorgt (s. a. USR).

Anweisung: NBGRD
Beispiel: NBGRD

Durch die NBGRD-Anweisung wird eine durch FCLS oder BGRD angewählte Hintergrundfarbe des FGR-Bildschirms wieder abgeschaltet (s. a. BGRD, FCLS n).

Anweisung: NEW
Beispiel: NEW

NEW löscht ein sich im Rechnerspeicher befindendes Programm.

Anweisung: NOT
Beispiel: IF NOT(A=4) THEN PRINT "A<>4"

NOT ist wie AND und OR ein logischer Operator. NOT verändert den logischen Ausdruck in einer IF-Anweisung. Es liefert den Wert eines Ausdrucks folgendermaßen modifiziert:
War der logische Ausdruck wahr, wird er durch NOT falsch.
War der logische Ausdruck falsch, wird er durch NOT wahr.

Anweisung: NPLOT x1,y1 (TO x2,y2)
Beispiel: NPLOT 10,10 (TO 20,20)

NPLOT löscht einen Punkt oder eine Linie auf dem FGR-Bildschirm, die zuvor durch eine PLOT-Anweisung gezeichnet wurde.

Anweisung: NSHAPE x,y
Beispiel: NSHAPE 10,10

Eine mit Hilfe der SHAPE-Anweisung auf dem FGR-Bildschirm erstellte Figur wird gelöscht.

Anweisung: ON ... GOSUB
Beispiel: ON X GOSUB 100,200,300

Ergebnisabhängiger Sprung zu Unterprogrammen.

Anweisung: ON ... GOTO
Beispiel: ON X GOTO 100,200,300

Ergebnisabhängiger Sprung.

Anweisung: OPEN (nur Disk-BASIC)
Beispiel: OPEN"0",1,"TEST"

Eröffnen einer Datei auf Diskette.

Anweisung: PAINT x,y,f1,f2
Beispiel: PAINT 10,10,3,2

PAINT ist eine Grafikanweisung zum farbigen Ausmalen von Flächen auf dem FGR-Bildschirm. Ausgemalt werden können nur in sich geschlossene Flächen. Beim Ausmalen einer Fläche sucht der Rechner beim Zeichnen jeder einzelnen Linie nach einer oder mehreren Begrenzungsfarben, bei denen das farbige Ausfüllen enden soll.

Anweisung: PEEK
Beispiel: PRINT PEEK(10000)

PEEK ist eine Anweisung zum Auslesen des Adreßinhalts einer bestimmten Speicherstelle.

Anweisung: PLAY
Beispiel: PLAY (1,4,1,15)

PLAY ist ein Befehl zur Erzeugung musikalischer Klänge über einen speziellen Baustein zur Tonerzeugung.

Anweisung: PLOT x1,y1 <TO x2,y2>
Beispiel: PLOT 10,10 TO 20,20

PLOT ist ein Grafikbefehl, mit dem Punkte oder Linien auf dem FGR-Bildschirm erstellt werden können (s. a. NPLOT, FCOLOUR).

Anweisung: POKE ad,w
Beispiel: POKE HF400,255

POKE ist ein Befehl zum Laden einer Speicheradresse ad mit einem Wert w (s. a. PEEK).

Anweisung: POS(x)
Beispiel: PRINT POS(0)

POS ermittelt die relative horizontale Position des Cursors auf dem Bildschirm. Das Argument x ist eine sog. „Dummyvariable“ und ohne Bedeutung, muß aber beim Aufruf der Funktion mit angegeben werden.

Anweisung: PRINT
Beispiel: PRINT“HALLO“

PRINT ist ein Befehl zur Ausgabe von Daten auf den Bildschirm, auf Kassette oder auf Diskette.

Anweisung: PRINT USING
Beispiel: PRINT USING"###.##DM";M

PRINT USING ist ein Befehl zur formatierten Ausgabe von Daten auf dem Bildschirm. Die hinter dem USING stehende Zeichenkette gibt an, in welcher Form die Ausgabe von Daten erfolgen soll.

Anweisung: PUT (nur Disk-BASIC)
Beispiel: PUT 1,4

In einer unter Disk-BASIC eröffente Direktzugriffsdatei wird durch PUT ein Datensatz mit einer vorgegebenen Satznummer abgelegt (s. a. GET).

Anweisung: RANDOM
Beispiel: RANDOM

Die Anweisung RANDOM bewirkt, daß das sich im Microsoft-BASIC befindende Programmelement, das für die Erzeugung von Zufallszahlen zuständig ist, mit neuen Werten versehen wird. Dadurch wird die „Zufälligkeit" von Zufallszahlen erhöht.

Anweisung: READ
Beispiel: READ W,W1

READ ist ein Einlesebefehl für Daten, die innerhalb eines Programms unter DATA-Anweisungen abgelegt sind.

Anweisung: REM
Beispiel: REM hier können Anmerkungen innerhalb des Programms stehen

Steht am Anfang einer Programmzeile die Anweisung REM oder das Zeichen <'>, wird alles, was dahinter steht, vom Rechner bei der Programmausführung überlesen.

Anweisung: RENUM
Beispiel: RENUM 10,1

RENUM ist ein Befehl zum Umnumerieren von Programmzeilen.

Anweisung: RESTORE
Beispiel: RESTORE

Die Anweisung RESTORE bewirkt, daß der Zeiger für die READ-Anweisung auf das erste Datenelement in der ersten DATA-Anweisung zurückgesetzt wird.

Anweisung: RESUME n
Beispiel: RESUME 10

Wurde nach Auftreten eines Programmfehlers durch eine ON ERROR GOTO-Routine zu einer bestimmten Zeilennummer verzweigt, kann der Rechner dort durch eine RESUME-Anweisung angewiesen werden, die Programmausführung ab einer bestimmten Zeilennummer n wiederaufzunehmen.

Anweisung: RETURN
Beispiel: RETURN

RETURN ist eine Anweisung an den Rechner, von einem Unterprogramm, zu dem durch eine GOSUB-Anweisung verzweigt wurde, wieder ins Hauptprogramm zurückverzweigen.

Anweisung: RIGHT$(S$,n)
Beispiel: PRINT RIGHT$("123",2)

Eine Zeichenkette S$ wird hinter dem n-ten Zeichen von rechts „abgeschnitten" (s. a. LEFT$, MID$).

Anweisung: RND(x)
Beispiel: PRINT RND(0),RND(100)

Die RND-Funktion dient zur Erzeugung einer Zufallszahl zwischen 0 und dem vorgegebenen Argument. Ist dieses ungleich 0, werden über den Zufallsgenerator ausschließlich ganzzahlige Werte erzeugt. Ist das Argument x gleich 0, werden Gleitkommazahlen ausgegeben, deren Werte zwischen 0 und 1 liegen.

Anweisung: SCALE n
Beispiel: SCALE 2

SCALE dient zur Festlegung des Größenfaktors, mit der eine in der Shapetabelle des Colour-Genie (Adresse 7F00H bei 16k-Geräten, Adresse BF00H bei 32k-Geräten) definierte Figur ausgegeben werden soll (s. a. SHAPE, NSHAPE).

Anweisung: SIN(x)
Beispiel: PRINT SIN(3.1415927/4)

SIN ist eine mathematische Funktion. Sie berechnet den Sinus eines Arguments, das im Bogenmaß vorgegeben werden muß (s. a. COS, TAN,ATN).

Anweisung: SOUND
Beispiel: SOUND 4,3

SOUND ist eine Anweisung zur Steuerung des Tongeneratorbausteins im Colour-Genie. Mit SOUND werden Spezialtoneffekte erzeugt.

Anweisung: SQR(x)
Beispiel: PRINT SQR(2)

SQR ist eine Funktion zur Berechnung der Quadratwurzel des durch x vorgegebenen Arguments.

Anweisung: STOP
Beispiel: IF Z=10 THEN STOP

Mit der STOP-Anweisung kann eine Unterbrechung eines Programms an einer vorbestimmten Stelle angegeben werden.

Anweisung: STRING$ (n,w)
Beispiel: PRINT STRING (30,65)

Durch die String-Anweisung wird eine Zeichenkette erzeugt, die n-mal das Zeichen mit dem ASCII-Code w enthält.

Anweisung: STR$(x)
Beispiel: A=STR$(4.5)

STR$ ist eine Konvertierungsfunktion. Aus dem Wert des Arguments x wird eine Zeichenkette gleichen Inhalts gebildet.

Anweisung: TAN(x)
Beispiel: PRINT TAN(3.1415927/4)

Bei Vorgabe eines Winkels x im Bogenmaß liefert die TAN-Funktion den Tangens des entsprechenden Winkels (s. a. SIN, COS, ATN).

Anweisung: TRON
Beispiel: TRON:RUN

Jede Programmzeile, die der Rechner bei Ausführung eines Programms durchläuft, wird auf dem Bildschirm protokolliert.

Anweisung: TROFF
Beispiel: TROFF

Die durch TRON eingeschaltete Protokollierung wird abgeschaltet (s. a. TRON).

Anweisung: USR(x)
Beispiel: X=USR(0)

Zu einer Maschinenunterroutine, deren Startadresse durch den Inhalt der beiden Adressen 16526 und 16527 vorgegeben ist, wird von BASIC aus verzweigt. Das Argument x wird dem HL-Registerpaar übergeben (s. a. NAME).

Anweisung: VAL(s$)
Beispiel: W=VAL("3.4")

VAL ist eine Konvertierungsfunktion. Eine Zeichenkette, deren Ziffernfolge eine Zahl repräsentiert, wird in ihr numerisches Äquivalent umgewandelt.

Anweisung: VARPTR
Beispiel: W=4:PRINT VARPTR(W)

Die VARPTR-Funktion liefert Informationen über Typ und Länge einer Variablen, sowie Hinweise auf die Adresse, wo die zu der Variablen gehörenden Daten im Speicher abgelegt wurden.

Stichwortverzeichnis

Die SYBEX Bibliothek

EINFÜHRUNG IN PASCAL UND UCSD/PASCAL
von Rodnay Zaks – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 535 Seiten, 130 Abbildungen, Ref.Nr.: **3004** (1982)

DAS PASCAL HANDBUCH
von Jacques Tiberghien – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen. 480 Seiten, 270 Abbildungen, Format 23 x 18 cm, Ref.Nr.: **3005** (1982)

PROGRAMMIERUNG DES Z80
von Rodnay Zaks – ein kompletter Lehrgang in der Programmierung des Z80 Mikroprozessors und eine gründliche Einführung in die Maschinensprache. 608 Seiten, 176 Abbildungen, Format DIN A5, Ref.Nr.: **3006** (1982)

PASCAL PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK
von Alan Miller – eine Sammlung von 60 der wichtigsten wissenschaftlichen Algorithmen samt Programmauflistung und Musterdurchlauf. Ein wichtiges Hilfsmittel für Pascal-Benutzer mit technischen Anwendungen. 398 Seiten, 120 Abbildungen, Format 23 x 18 cm, Ref.Nr.: **3007** (1982)

BASIC COMPUTER SPIELE/Band 1
herausgegeben von David H. Ahl – die besten Mikrocomputerspiele aus der Zeitschrift „Creative Computing“ in deutscher Fassung mit Probelauf und Programmlisting. 208 Seiten, 56 Abbildungen, Ref.Nr. **3009**

BASIC COMPUTER SPIELE/Band 2
herausgegeben von David H. Ahl – 84 weitere Mikrocomputerspiele aus „Creative Computing“. Alle in Microsoft-BASIC geschrieben mit Listing und Probelauf. 224 Seiten, 61 Abbildungen, Ref.Nr.: **3010**

VORSICHT! Computer brauchen Pflege
von Rodnay Zaks – das Buch, das Ihnen die Handhabung eines Computersystems erklärt – vor allem, was Sie damit nicht machen sollten. Allgemeingültige Regeln für die pflegliche Behandlung Ihres Systems. 240 Seiten, 96 Abbildungen, Best.-Nr.: **3013** (1983)

BASIC PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK
von Alan Miller – eine Bibliothek von Programmen zu den wichtigsten Problemlösungen mit numerischen Verfahren, alle in BASIC geschrieben, mit Musterlauf und Programmlisting. 352 Seiten, 147 Abbildungen, Best.-Nr.: **3015** (1983)

EINFÜHRUNG IN DIE TEXTVERARBEITUNG
von Hal Glatzer – woraus eine Textverarbeitungsanlage besteht, wie man sie nutzen kann und wozu sie fähig ist. Beispiele verschiedener Anwendungen und Kriterien für den Kauf eines Systems. 248 Seiten, 67 Abbildungen, Best.-Nr. **3018** (1983)

EINFÜHRUNG IN WORDSTAR
von Arthur Naiman – eine klar gegliederte Einführung, die aufzeigt, wie das Textbearbeitungsprogramm WORDSTAR funktioniert, was man damit tun kann und wie es eingesetzt wird. 240 Seiten, 36 Abbildungen, Best.-Nr.: **3019** (1983)

PLANEN UND ENTSCHEIDEN MIT BASIC
von X. T. Bui – eine Sammlung von interaktiven, kommerziell-orientierten BASIC-Programmen für Management- und Planungsentscheidungen. 200 Seiten, 53 Abbildungen, Best.-Nr.: **3025** (1983)

BASIC FÜR DEN KAUFMANN
von D. Hergert – das BASIC-Buch für Studenten und Praktiker im kaufmännischen Bereich. Enthält Anwendungsbeispiele für Verkaufs- und Finanzberichte, Grafiken, Abschreibungen u.v.m. 208 Seiten, 85 Abbildungen, Best.-Nr.: **3026** (1983)

ERFOLG MIT VisiCalc
von D. Hergert – umfassende Einführung in VisiCalc und seine Anwendung. Zeigt Ihnen u. a.: Aufstellung eines Verteilungsbogens, Benutzung von VisiCalc-Formeln, Verwendung der DIF-Datei-Funktion. 224 Seiten, 58 Abbildungen, Best.-Nr.: **3030** (1983)

PLANEN, KALKULIEREN, KONTROLLIEREN MIT BASIC-TASCHENRECHNERN
von P. Ickenroth – präsentiert eine Reihe von direkt anwendbaren BASIC-Programmen für zahlreiche kaufmännische Berechnungen mit Ihrem BASIC-Taschenrechner. 144 Seiten, 48 Abbildungen, Best.-Nr.: **3032** (1983)

MEIN ERSTES BASIC PROGRAMM
von Rodnay Zaks – das Buch für Einsteiger! Viele farbige Illustrationen und leichtverständliche Diagramme bringen Spaß am Lernen. In wenigen Stunden schreiben Sie Ihr erstes nützliches Programm. 208 Seiten, illustriert, Best.-Nr.: **3033** (1983)

Z80 ANWENDUNGEN
von J. W. Coffron – vermittelt alle nötigen Anweisungen, um Peripherie-Bausteine mit dem Z80 zu steuern und individuelle Hardware-Lösungen zu realisieren. Ca. 320 Seiten, ca. 200 Abbildungen, Best.-Nr.: **3037** (September 1984)

COMMODORE 64 – LEICHT GEMACHT
von J. Kascmer – führt Sie schnell in die Bedienung von Tastatur, Bildschirm und Diskettenlaufwerken ein, macht Sie zum BASIC-Programmierer Ihres C64! 176 Seiten, 36 Abbildungen, Best.-Nr.: **3038** (1984)

SPIELEN, LERNEN, ARBEITEN mit dem TI99/4A
von K.-J. Schmidt und G.-P. Raabe – eine eingehende Einführung in die Bedienung und Programmierung des TI99/4A. Mit den vielen Beispielprogrammen holen Sie das Beste aus Ihrem Computer heraus. 192 Seiten, 41 Abbildungen, Best.-Nr.: **3039** (1984)

MEIN ERSTER COMPUTER
von Rodnay Zaks – Der unentbehrliche Wegweiser für jeden, der den Kauf oder den Gebrauch eines Mikrocomputers erwägt, das Standardwerk in 3., überarbeiteter Ausgabe. 304 Seiten, 150 Abbildungen, zahlreiche Illustrationen, Best.-Nr.: **3040** (1984)

ERFOLG MIT MULTIPLAN
von Th. Ritter – das Tabellenkalkulations-Programm Multiplan hilft Ihnen bei der Lösung kommerzieller, wissenschaftlicher und allgemeiner Probleme. Lernen Sie die Möglichkeiten kennen, Ihre Software optimal zu nutzen! 208 Seiten, ca. 60 Abbildungen, Best.-Nr.: **3043** (1984)

FARBSPIELE MIT DEM COMMODORE 64
von W. Black und M. Richter – 20 herrliche Farbspiele für Ihren C64, mit Beschreibung, Programmlisten und Bildschirm-Darstellungen. Für mehr Freizeit-Spaß mit Ihrem Commodore! 176 Seiten, 58 Abbildungen, Best.-Nr.: **3044** (1984)

COMMODORE 64 BASIC HANDBUCH
von D. Hergert – zeigt Ihnen alle Anwendungsmöglichkeiten Ihres C64 und beschreibt das vollständige BASIC-Vokabular anhand von praktischen Beispielen. 208 Seiten, 92 Abbildungen, Best.-Nr.: **3048** (1984)

COMMODORE 64 PROGRAMMSAMMLUNG
von S. R. Trost – mehr als 70 getestete Anwenderprogramme, die direkt eingegeben werden können. Erläuterungen gewährleisten eine optimale Nutzung. 192 Seiten, 160 Abbildungen, Best.-Nr.: **3051** (1983)

CP/M-HANDBUCH
von Rodnay Zaks – das Standardwerk über CP/M, das meistgebrauchte Betriebssystem für Mikrocomputer. Für Anfänger eine verständliche Einführung, für Fortgeschrittene ein umfassendes Nachschlagewerk über die CP/M-Versionen 2.2, 3.0 und CCP/M-86 sowie MP/M. 2., überarbeitete Ausgabe. 320 Seiten, 56 Abbildungen, Best.-Nr.: **3053** (1984)

MEIN ERSTES COMMODORE 64-PROGRAMM
von R. Zaks – sollte Ihr erstes Buch zum Commodore 64 sein. Viel Spaß am Lernen durch farbige Illustrationen und leichtverständliche Diagramme, Programmieren mit sofortigen Resultaten. 208 Seiten, illustriert, Best.-Nr.·**3062** (1984)

SYBEX MIKROCOMPUTER LEXIKON
– die schnelle Informationsbörse! Über 1500 Definitionen, Kurzformeln, Begriffsschema der Mikroprozessor-Technik, englisch/deutsches und französisch/deutsches Wörterbuch, Bezugsquellen. 192 Seiten, Format 12,5 x 18 cm, Best.-Nr.: **3035** (1984)

FORDERN SIE EIN GESAMTVERZEICHNIS UNSERER VERLAGSPRODUKTION AN:



SYBEX-VERLAG GmbH
Vogelsanger Weg 111
4000 Düsseldorf 30
Tel.: (02 11) 62 64 41
Telex: 8 588 163

SYBEX
6–8, Impasse du Curé
75018 Paris
Tel.: 1/203-95-95
Telex: 211.801 f

SYBEX INC.
2344 Sixth Street
Berkeley, CA 94710, USA
Tel.: (415) 848-8233
Telex: 336311

SYBEX

Spiele in Sound und Farbe –
der Freizeit-Spaß mit einem Computer,
der vieles kann!

Mein Colour Genie

Es gibt für den Colour Genie bereits eine Vielzahl von Spielprogrammen. Wenn Sie aber eigene Programme für diesen Computer entwickeln möchten, bekommen Sie durch dieses Buch eine Fülle von Anregungen.

Es zeigt,

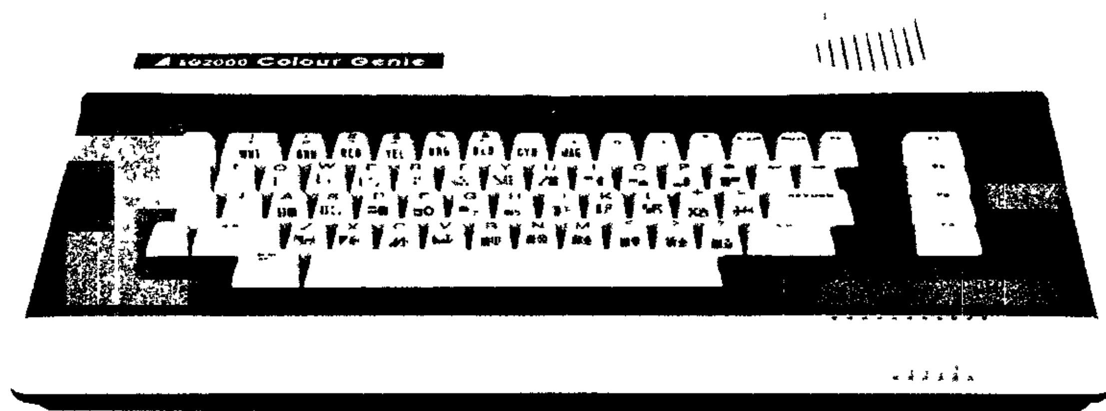
- mit welchen Befehlen der Zeichensatz geändert werden kann,
- wie Sie eigene Funktionen entwickeln und benutzen können,
- was alles mit dem Soundgenerator gemacht werden kann,
- was bei der Verwendung von Diskettenlaufwerken beachtet werden muß,
- wie Sie zum Beispiel „Reversi“ auf dem Colour Genie programmieren können.

Der Autor vermittelt Ihnen viele praktische Beispiele, die leicht nachzuvollziehen sind. Sie lernen, mit einfachen Mitteln eindrucksvolle Programme zu erstellen und so das Beste aus Ihrem Colour Genie herauszuholen.

ISBN 3-88745-063-9

SYBEX

Colour Genie



Ralf Marquis





Spiele in Sound und Farbe – der Freizeit-Spaß mit einem Computer, der vieles kann!

Mein Colour Genie

Es gibt für den Colour Genie bereits eine Vielzahl von Spielprogrammen. Wenn Sie aber eigene Programme für diesen Computer entwickeln möchten, bekommen Sie durch dieses Buch eine Fülle von Anregungen.

Es zeigt,

- mit welchen Befehlen der Zeichensatz geändert werden kann,
- wie Sie eigene Funktionen entwickeln und benutzen können,
- was alles mit dem Soundgenerator gemacht werden kann,
- was bei der Verwendung von Diskettenlaufwerken beachtet werden muß,
- wie Sie zum Beispiel „Reversi" auf dem Colour Genie programmieren können.

Der Autor vermittelt Ihnen viele praktische Beispiele, die leicht nachzuvollziehen sind. Sie lernen, mit einfachen Mitteln eindrucksvolle Programme zu erstellen und so das Beste aus Ihrem Colour Genie herauszuholen.

ISBN 3-88745-063-9